IEEE P2000.2
Draft Recommended Practice for Information Technology
Year 2000 Test Methods
Draft 12

IEEE Std P2000.2 was prepared by the P2000.2 Working group of the IEEE Portable Applications Standards Committee.

Introduction

This Test Method's recommended practice provides the framework for detailed planning and execution of all steps and tasks involved in testing for Year 2000 compliance. The resulting test plan would outline the testing approach and identify system elements that are at risk of failure when crossing into the Year 2000 or using data that includes dates after 2000-01-01. The focus of this Test Methods Recommended Practice is to ease the transition for companies through Year 2000 testing by outlining a range of example test cases to aid and expand the capabilities of the individual companies to assess, test, and verify software, hardware and firmware. This recommended practice should be used with IEEE Std 2000.1-1998 (Year 2000 Terminology).

This recommended practice describes the process, test methods, and remediation for Year 2000 assessment and validation. It is intended for individuals or organizations that develop, test, acquire, or use software, firmware, or hardware.

This recommended practice is designed to help individuals and organizations: This document does not attempt to develop a comprehensive test suite. It is expected that systems affected by Year 2000 related problems will range from single chip embedded systems to global networks. Since test procedures are typically system dependent, it is impossible to address every Year 2000 problem. Instead, included are examples of sample test cases that are intended to spur investigation into similar functions in a specific application. These sample test cases were intended to be generally applicable. Furthermore, the test cases are intended to be *problem* specific and not application specific and may have applicability to a wide variety of applications or situations. For this reason application-specific terminology has been avoided where possible. It is likely that a specific system may need hundreds of test cases related to a single type of module. The process of writing the test case steps and results will be facilitated using the sample test cases. It should be noted at this point that this *recommended practice* guide focuses only on the century digit change problem, it does not address date counter overflow.

Warning: Testing for Year 2000 related issues on operational systems might cause damage to the system or the data contained therein. Some of these issues will be further addressed in this document.

Organization of recommended practice

The standard is divided into six sections:

- Statement of Scope, and conformance information (Section 1)
- Normative References, list of normative references (Section 2)
- Definitions of terms as used in this document (Section 3)
- General requirements, editorial comments and warnings (Section 4)
- Methodology for creation of a test plan (Section 5)
- System elements at risk – Example test cases (Section 6)
- Informative Annexes

IEEE Std P2000.2 Recommended Practice for Information Technologies Year 2000 Test Methods was prepared by the P2000.2 Working Group of the IEEE Computer Society. At the time this best practice was approved, the membership of the IEEE PASC working group was as follows:

Participants

At the time IEEE completed this document the Year 2000 Test Methodology Working Group had the following membership:

Chairperson

Lowell Johnson

Technical editor

Christina Drukala

Major contributors

Steven Brock
Christina Drukala
Don Estes
Vincent E. Henley
Thomas P. Koenig
John Napier
Alan Peltzman
Terrill J. Slocum

Work group

Steven Brock
Johnathon Chapman
Eldon Colby
David Dodd
Christina Drukala
Don Estes
Vincent E. Henley
Dr. John S. Davies
Nelson deGrandmaison
Victor Grebler
Mary Hengstebeck
Thomas P. Koenig
Bob Lynch
John Napier
Alan Peltzman
John Rusell
Terrill J. Slocum
Gary R. Young

This list may be incomplete at any time prior to commencement of balloting

202

IEEE Recommended Practice for Information Technology - PASC Year 2000 Test Methods

1 Overview

The Test Method Recommended Practice is designed to assist organizations and individuals in both the assessment of their assets for Year 2000 status and validation of their Year 2000 readiness after assets have been remediated. The problem domain that the technology community is facing is how to assess system elements within their organization that may be at risk of failure due to Year 2000 related problems. This problem may affect software, firmware, hardware and data elements. The private enterprise, government agencies and the military all face the possibility of failures. Because the problem is pervasive, this recommended practice will not focus on any particular platform, technology or industry. This Recommended Practice is not a certification. It is a template that can be customized by each organization to help in the creation of Year 2000 Test Plans and Test Scenarios for the technical assessment, verification and validation of their individual assets. Each organization will need to prioritize and apply good Engineering Test Methodology when executing Year 2000 testing activities.

The procedures in this document are not intended as a certification. This document's intent is to serve as an outline to help guide personnel responsible for testing Year 2000 problems and isolate troubled areas found during Year 2000 remediation. Subsequent sections provide a non-exhaustive list of system elements and components that may have difficulty transitioning into the Year 2000. The list should be compared against the functions and procedures used by the system, similar system elements should be tested. The system elements at-risk and the sample test assertions are to be used in cooperation to enable a testing organization to create customized test assertions for specific assets, whether software, firmware, hardware and data elements.

This Recommended Practice is divided into six areas. The first area contains information on the scope and purpose of the document. The second section lists supporting information necessary for implementation of the procedures in this document. The third section contains definitions of terms used within this document. Section 3 gives a description of the test case outline used in Section 5 and provides important warnings. Section 5 supplies guidance on methodology, process, and testing impact of different remediations, validation and reporting for testing of Year 2000. In section six there are example system elements and features that have a high probability of being affected by date based algorithms and therefore could fail if not designed to operate on both sides the 1999 to Year 2000 boundary. This area also contains example test assertions that are generic in nature and correlate with system elements. Section 7 is annex data that may be valuable, but is provided for informational purposes only.

1.1 Scope:

The scope of this document is to identify Recommended Practices for defining a user-specific test plan for Year 2000 validation. They include:

- A taxonomy of system elements and features which are likely to exhibit Year 2000 failures,
- Test scenarios which may be used as templates for the creation of test cases to detect defective system elements with respect to the Year 2000, and

- A customizable methodology and process for determining the testing impact of various remediation techniques, validation techniques, and documentation of test results.

Significant engineering judgement must be exercised by the user of this document in applying its recommendations; it is not intended that these practices be employed in a mechanical or rote fashion. It should be used to supplement standard testing procedures that encourage proper coverage and consistency of testing.

1.2 Purpose

This document is to provide users of computer hardware, firmware, software or data. Systems with Recommended Practices for assessing and demonstrating the system element's within their organization that may be at-risk of failure due to the Year 2000 problem and related date-specific issues.

1.3 Assumptions

The examples and test cases used throughout this document assume the following conditions are true unless otherwise stated:

- Existing systems are complete, stable, and tested and all features work according to a specification, documentation or a functional baseline or a specific agreement among the parties.
- Once changes to the code have been made to correct Year 2000 problems, a regression test or appropriate function test is executed to validate that system functionality including impacted date functions are working correctly.
- The system element's date/time formats are convertible to the Gregorian Calendar for the relevant range of dates over which the test cases will run.
- Testing is performed in an environment that will not damage or impact real-world data. (See the sub-clause **4.2 Warnings**.)
- The test facility has the ability to change the time clock of the system or simulate its change and in some cases age data.

2 Normative references

The following standards contain provisions that, through references in this text, constitute provisions of this recommended practice. This recommended practice shall be used in conjunction with the following standards.

{1} IEEE Standard 2000.1 1998 Information technology – Year 2000 Terminology

{2} ISO 8601:1988 Data Elements and Interchange formats – Information exchange – Representation of Dates and Times

{3} IEEE Standard 100 1997

3 Definitions

For the purposes of this recommended practice, the following terms and definitions apply:

3.1 acceptable deviation

In the context of evaluating specific test case post conditions, a deviation is permitted based on an informed decision to specify that deviation as non-critical.

3.2 bridge

A bridge is a system element which converts from one data format to another. Bridging can be incorporated in most Year 2000 remediation to interpret date-data formats. This may be helpful in transferring dates between date formats for remediated system elements and those used in the original system. There may also be situations in which multiple remediation techniques requiring different date formats are used, creating a need for bridges between them.

3.3 failure

Any deviation from specified post condition of a test case is considered a failure for that specific test case. Post conditions should require adherence to the specification, documentation or functional baseline for the system.

3.4 future regression

Regression testing provides a quick way to test broad areas of a system's functionality. Future regression testing expands the normal regression process to include future data, and advanced system dates.

3.5 Gregorian calendar (G)

The revision of the Julian calendar in 1582 by Pope Gregory XIII. Since the earth orbits the sun in an average of 365.24219 days not 365.25 days the Julian Calendar gradually fell out of sync with astronomical events used to set the dates of some holidays. To compensate it was decreed that Thursday, October 4, 1582 would be directly followed by Friday, October 15, 1582 in the Gregorian calendar. The elimination of the 10 days would affect calculations that spanned that period. The rules governing the occurrence of leap year were modified to adjust the calendar to more closely reflect the celestial year. In the Gregorian Calendar a year is a leap year if it is evenly divisible by 4 unless it is also divisible by 100. In addition any year evenly divisible by 400 is considered a leap year. The Gregorian Calendar was gradually adopted by most western nations between 1582 and 1752, and has remained in use to the present.

3.6 Julian calendar (J)

Introduced in Rome in 46 BC, the Julian Calendar established a 12-month year of 365 days with every 4th year having 366 days resulting in an average year length of 365.25 days.

3.7 Julian date (JD)

The term Julian date has been used for a real number representing the number of days from a specified start date in many systems. Officially a Julian date is a

representation in which a real number specifies the number of days and fractions of days since noon of January 1, -4712J (the equivalent Gregorian date/time is noon on November 24, -4713). Some confusion occurs because of the existence of a modified Julian date (MJD) with a start point of November 17, 1858G. The documentation of some systems use the term 'Julian Date' to refer to either an ordinal date or a value equal to the number of days from a start date specific to the system. When Julian dates are used in a system it is best to research the system specific meaning of the term.

3.8 Lilian day number (LDN)

A Lilian Day Number (LDN) is defined as the number of days since October 14, 1582 on the Gregorian calendar. Confusion arises because October 14, 1582 is skipped in the Gregorian calendar. LDN 0 therefore has no equivalent Gregorian date. LDN 1 falls on October 15, 1582. For other non- integer date numbering systems see 2.11 Relative Integer Dates (RID) below. As with Julian dates it is best to research the system specific definition of the term 'Lilian Date'.

3.9 ordinal date (OD)

A form of date notation that consists of a year value and an integer value indicating the number of days from the beginning of the year. For example 1997-1-13 is 1997-013 in ordinal notation. The day value ranges from 1 to 365 normally and 1 to 366 in leap years.

3.10 pass

The lack of any deviation from the expected post condition of a test case signifies a pass for that specific test case. Adherence to specification or documentation or functional baseline indicates a pass.

3.11 real time clock (RTC)

A hardware system element that provides the system with a reference to real world time. A common implementation would be a circuit containing a set of registers holding the current month, day, year, day-of-week number and other time related values along with a circuit that continuously updates these register values. The circuit is normally provided with an alternate power source such as a battery that allows the RTC to continue to function when main system power is not available.

3.12 special logic

Many programs use specific dates to trigger exceptions to normal date processing. A common example is expiration date on tape archives. Rather than adding another flag to the tape header the date 1999-9-9 is used by many systems to mean never expire. Other dates have been used to indicate an unknown or out of range date. Since there is no standard for this practice, the specific dates used may be difficult to trace. **The potential for dramatic failures increases in 1999 due to Date Code Flags and various other Year 2000 problems**

3.13 system elements

A system is a collection of components organized to accomplish a specific function or set of functions. In this document, system elements refer to any Software, Hardware or Firmware components or combination of them that perform a specified task.

3.14 system time

The state of any system element that is used to synchronize system events with real world events based on a date, time or combination of the two. System time is usually maintained in a hardware system element that is protected against unintentional changes to its value that might be caused by a system reset or a local power outage.

This standard refers to the time source as a Real Time Clock (RTC) to distinguish it from the system clock that synchronizes the internal processor functions. The RTC may be read directly, but more often its contents is stored in another clock register that is incremented separately from the RTC. The two times may be synchronized at specified intervals or only when the system is started. An example is the X86 based PC that has a battery backed RTC. At start up the RTC is used to determine the date and time. These values are reformatted and placed in memory. From then on the Time and date are incremented by the System BIOS. Some operating systems may maintain another clock that is updated by the operating system itself. Changing the system time requires resynchronization of the clocks.

Distributed systems, including networks and plant automation systems may have multiple RTCs residing in many individual components. The concept of system time assumes that these clocks are synchronous or nearly synchronous. This may be accomplished by automatically setting each individual RTC to the value of a designated master RTC. In Asynchronous systems individual RTCs may be set manually, introducing a error to the system time that might be several minutes.

Some systems do not have, or need a system time. This may be true of Countdown timers, calculators, simple logical controllers and other systems or system elements that perform in exactly the same manner regardless of the date or time. Caution should be used in identifying these systems since there is a possibility that a timer based on Gregorian dates may be used to implement timing functions even though there is no external reference to the date.

3.15 user defined date systems

Some users may implement locally developed date systems used to track user specific events. For example manufacturing, production or Just–In-Time (JIT) dates. Methodologies developed for verification of Year 2000 issues associated with the Gregorian calendar may need to be adapted to apply to user defined date systems.

3.16 Year 2000 rollover

The instant when a system's year changes from 1999 to 2000. In a system that uses a six-character date format this is a transition from 99-12-31 (YY-MM-DD) to 00-01-01 (YY-MM-DD).

4 General requirements and warnings

4.1 Editorial conventions

4.1.1 Test case format

The sample test cases included in this Recommended Practice have been made as general as possible. Every effort has been made not to rely on the feature set of a specific system. In many situations the actual process necessary to execute a step may be much more complicated than indicated in these test cases.

Test cases are developed to satisfy various test objectives. Information elements common to a test or test case include Test Scenarios, Test objectives, Test conditions, Test procedures, and Expected test results:

4.1.1.1 Test scenarios

Most test cases are proceeded by a paragraph explaining the choices made in the test design and their intended results. In many cases the test data or procedures will need to be modified in order to fit a specific situation. The initial paragraph is intended as an aid to understanding the purpose of the test case so that it can be used as a model for other test case designs.

4.1.1.2 Test objective

This is a one or two line statement of the purpose of the test case.

4.1.1.3 Test conditions

Settings that need to be modified from normal operating parameters are noted in this area.

4.1.1.4 Test procedures/results

Procedures are actions necessary to cause an observable result. The test format used here presents the procedure, the result, and a step number in a table to enhance readability. In some cases several actions may be required to produce a single result. In these instances the result column is left blank until the step where the final action in the sequence is completed.

4.1.1.5 Test dates – holidays

Holidays vary by jurisdictions in terms of both legal and cultural context. However, these may require specific action related to pay-rates, intrusion alarms, or secure area access. Examples of holidays are used in this document, mostly referencing U.S. “National” holidays. Users of this document should be aware of the diversity and variation of holidays where their operations are concerned, and adapt these examples for these contexts.

4.2 Warnings

The warning section is designed to highlight potential areas of risk resulting from testing. This section may provide suggestions on how to avoid loss or damage to system or system elements. Ensure that a disaster recovery plan is up to date and ***operational*** for restoration of production data in case of loss. Fully outlining a process for creating a disaster recovery plan is outside the scope of this document.

As noted in section **5.2.4 Build test environment,** it is best to conduct all tests on a duplicate system when possible.

Complete backups of all corruptible data storage devices are essential. Consider maintaining a second set of media (Hard disks on sliding brackets, Disk Packs, tapes, etc.) with the unremediated system installed on them.
Embedded systems may pose special safety problems both because they often operate autonomously and because they may control systems with the potential to do physical harm to equipment, personnel or to the public at large.

One way to alleviate these risks is to remove system control elements from the larger system. Disconnecting them from physical outputs and connecting them to a device capable of displaying the output state of the controller. These devices can be as simple as a multi-meter or breakout-box. Systems with many control outputs or critical interactions between input sensors and outputs may require software simulations. A simulation can be programmed to respond to the system elements control outputs simulate the timing or function of mechanical devices and actuate the elements sensor inputs.

While remediation of the system is sometimes done entirely in a software simulation the verification and validation of the system elements should be done using a hardware interface that allows the actual hardware component to be plugged into the simulation.

Simulation can be used to detect the majority of errors before the embedded controller is tested in the full system. However, no simulation is perfect. Discrepancies between the simulation and the real system can cause failures. Careful real world testing should be conducted after testing within the simulation is completed.

4.2.1 Automated purge: Setting the system date to a future date could trigger purge routines causing data loss.

4.2.2 Data Integrity: Ensure that data integrity is maintained during reintegration of system elements after testing/validation.

4.2.3 Data loss: Adequate backups should be made prior to execution of tests. The potential for dramatic loss of data increases in 1999, due to date code flags and various other Year 2000 problems.

4.2.4 Hardware damage: Use of future dates in system elements could damage hardware in embedded systems. For example some system elements have maintenance triggers that force the system to fuse hardware components, if not maintained within the periodic maintenance schedule causing physical damage to hardware.

4.2.5 Security access: Ensure that security accesses are not lost after manipulating dates on system. Changing a system date to a future date may result to the loss of some or all access codes to the system. In some systems, once a date is advanced for test purposes, it becomes impossible to retract the date without a full system reinstallation.

4.2.6 Software license expiration: Software with enforced licensing cannot be tested with the system clock set beyond the expiration. The software vendor should be contacted to arrange special licensing privileges.

4.2.7 Software license violation: Making a testing environment by installing copies of an application could violate licensing agreements and copyrights. This may trigger electronic Licensing Managers to restrict active concurrent copies of the application.

4.2.8 System corruption: Systems whose dates are set to the future may suffer operating system corruption upon reset to present date. For example system logs may be corrupted with invalid system time values. In cases like these, restoration or reinstallation of the operating system may be necessary.

4.2.9 Systems integration: Incomplete or incompatible remediation techniques within a larger system can lead to system or data corruption, either during testing or during reintegration. For example, incompatible date formats can lead to corrupted data.

4.2.10 Compiler pre-processors: Extra Caution may be warranted for compiler pre-processors where the date retrieval mechanism may differ from and override that of the compiler.

5 Methodology

This section outlines the Year 2000 Life cycle, and details the test methodology as it applies to the Year 2000 problem, and the use of test methodologies for assessing and testing the systems elements at risk.

5.1 Year 2000 life cycle:

A conversion model comprised of five phases each representing a major Year 2000 activity. Both the private and public sectors have used this model in addressing their respective Year 2000 issues. The five phases are described below:

- *Planning and awareness:* Define the Year 2000 problem and gain executive level support and sponsorship for establishing the problem as a high priority item for resolution. Research and establish a project plan, and obtain budget and resources. Note that the planning activities are also relevant to the other phases described below.

- *Assessment (inventory):* Evaluate the Year 2000 impact on the enterprise; develop contingency plans to handle data exchange issues and system failures (dysfunction or system crashes); prioritize systems by identifying those that are mission-critical.

- *Remediation (renovation):* Convert, replace, eliminate, work around, or encapsulate one or more system elements; modify interfaces.

- *Validation (audit):* Test, certify, and validate all system elements that have been converted or replaced

- *Implementation:* Place into production all system elements that have been converted or replaced

This Recommended Practice does not address the subject of validation within the Implementation Phase, which is generally outside the scope of testing/validation organizations. However, due to potential interdependencies between any remediated/validated system and the operational environment within which it must function, there is a likelihood that additional problems will emerge in the Implementation Phase. The origins of these problems could be in the remediated/validated system, in some other system(s) in the environment, or in the interactions among them. Therefore, it is recommended that an effort be made to plan, audit and manage the Implementation Phase with the goal of detecting, locating and addressing such problems.

5.2 Test process

Experience with Year 2000 projects indicates that a major portion of the effort is taken up by testing. Limited resources, high cost, and restricted project duration make it essential that organizations and individuals use an efficient and effective testing process.

5.2.1 Establish a test strategy:

It is key to success that the testing strategy be largely driven by business, not just technical considerations. It is important to ensure that there is no disconnect between those responsible for the general management of the business and those responsible for its Information Technology. Neither party in this partnership can arrive at the optimized testing strategy for the enterprise alone. Technical managers have a responsibility to ensure that business managers have the most current and accurate information on which to base key decisions. Business managers have a responsibility to understand the risks described in technical terms and to ensure that appropriate funding and resource decisions are made to achieve the best possible outcome for the enterprise.

A testing strategy is top-level guidance concerning the nature, constraints and desired outcomes of a testing effort. It guides the development of all the specific test activities that follow. Elements of a strategy can result from testing impacts of the type of system or system elements being tested. Factors critical to the success of the testing effort are identified. Level of risk and appropriate level of testing effort are identified. Test scope and who performs testing is determined. Tradeoffs are identified between competing planning factors such as schedule, cost, scope, technological approaches and quality.

Many organizations may find that there are not enough resources or time to test all their applications for Year 2000 compliance in a conventional or comprehensive manner. For those organizations, a prioritization of business functions and their supporting systems should be determined. The strategic priority is based on the risk to the enterprise should systems fail. This prioritization results in a series of risk levels being defined that can be applied to an organization's systems. These risk levels determine the level of testing effort required for each system. An example of such a risk hierarchy is presented in Annex - D Alternative testing methodology (informative).

It is appropriate to understand from the outset what the available time, human and machine resource constraints are. The test strategy and subsequent test plans must fit within the envelope of what can be done. It is inappropriate and futile to create plans and strategies that cannot be executed within the available constraints. It is strongly recommended that determining this envelope of constraints be the first order of business before elaborate, but unusable plans are developed.

The level of testing effort and criteria for completion for a system depends on the strategy, risk assessment and remediation technique selected. Completion criteria are often expressed as the degree of functional and/or code coverage achieved by the testing effort. Testing effort can range from the decision not to test through exhaustive testing. Testing effort can have a direct correlation to the risk of operational safety.

Year 2000 testing also requires that the differences between system times and asynchronous data times are understood and that appropriate date intervals are used that are sufficient and appropriate for the system being tested. System time is determined by clock mechanisms on the system being tested. The system clock provides the increment of time that synchronizes time date-data processing information for both function-specific purposes and system operational requirements such as internal logs or file time stamping. Data time has to do with the dates used within the data being processed, which can be current data or data aged beyond 2000 to reflect a future functional need. Testing of both system and data time is necessary, but the dates selected should be appropriate for the system under test. For example, the valid date interval for a strategic planning system could be from the present to the year 2030. The valid date interval for an income tax computation system could be a year or less.

5.2.2 Set test objectives

Test objectives are an identified set of system elements to be measured under specified conditions by comparing date related behavior with the expected behavior. The process of establishing Year 2000 test objectives may include a review of the test strategy and system requirements. Considering such factors as size, complexity, environment and reliability can be helpful in assessing risk associated with the system elements. Testing objectives are defined to address portions of system elements that have high risk and completion criteria should also be noted. The test completion criteria must be clearly defined and measurable.

5.2.3 Develop a test plan

The test plan is a document describing the scope, approach, resources and schedule of intended test activities. It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning. The test plan should clearly specify the metric that defines the end point of the test so that all concerned parties can agree when the test is complete. There are many different types of metrics. For example, number of test cases run successfully, defect discovery rate threshold, or mean time to failure.

The Test Plan should be appropriate to the test strategy, objectives and risk. Examples of levels of risk and the associated levels of testing effort are presented in **Annex - D Alternative testing methodology (informative)**.

5.2.3.1 Define tests

The full range of system functions that depend on the remediated system elements are tested to insure that any changes do not adversely affect related functions.
Tests address the following considerations in order to satisfy the test objectives:

- **Functional attributes.** Determine the functionality of the system elements impacted by date variables. Determine the expected outputs from the operational logic that acts on these variables. This can be approached from a data-oriented input versus output perspective (black box) or from a logic perspective (white box). In order to meet the testing criteria established for the system, it may be necessary to analyze data structures, code logic or the use of reserved dates or special dates (e.g., quarter end, month end) in the code or operational logic.

  The use of aged data is a necessary but not sufficient condition for successful Year 2000 testing. Both data, which is aged statically, that is, converted permanently to an aged condition, and dynamically, converted in process, may be needed. Changing the system clock does not age the data.

  Aged data are data that have been modified such that date variables within that data are changed from their current value to some future date. Normally, the year value must be changed to be useful in a Year 2000 testing context. Care must be taken to understand such things affected by the change of year, such as day of the week and Leap Year, and to either ensure consistency or to account for that difference in any testing scenario. An alternative to modifying existing data to an aged state is to create test data which has the same aged characteristics as modified existing or production data. This data may be data that has no other reason for existing other than Year 2000 testing.
- **Non-functional attributes**. Evaluate the structural attributes of the system under test to identify those attributes which may require date-related testing that are not dependent on the system functionality. Such date-related factors include performance, usability, maintainability, reliability, availability, serviceability,

portability, extendibility, and security. For example, the use of complex windowing remediation techniques may reduce the readability of the program code. Portability may be a concern if program code is changed on one platform and compiled or executed on another.

- **System time, data time and valid date intervals**. As noted in the discussion of strategy, testing of both system and asynchronous data time is necessary, but the dates selected should be appropriate for the system under test.

- **Remediation technique impacts.** Testing impacts of various Year 2000 remediation techniques are described in sub-clause **5.4 Test Impact of Remediation Techniques**.

- **Special-condition dates.** Consider the special-condition dates in sub-clause **5.7 Special date conditions.**

- **Existing test cases.** Evaluate existing tests and test data. Some of the needed tests may have already been developed during normal development testing or the assessment phase.

- **System elements at risk.** Review the sample test cases to identify elements at-risk and to assist with the creation of tests appropriate to the system. See clause **6 System elements at risk.**

5.2.3.2 Develop test cases

Once tests have been defined, test cases should be developed to fully exercise the requirements of the test. Information found within a test case typically includes test objective, test conditions, test procedures, and expected test results. An IEEE standard for test documentation is listed in **Annex - F Informative references (informative).**

As test cases are developed, other components necessary to perform the tests should be defined. For example, test sequencing, test procedures, testing tools and sources of test data are identified. Tool requirements can include file comparison tools, date simulation tools, date-data aging tools, transaction capture tools, and tools for logical test path analysis.

5.2.4 Build test environment

The next step of the process is to create the necessary environment to support testing. For example develop test simulations, test automation and test data.

Creating the test environment requires care because of the dangers listed in sub-clause **4.2 Warnings**.

Critical systems that must remain in operation during remediation and testing may require that an isolated hardware environment be used. A baseline should be created for this system to ensure that the unremediated test system accurately represents the original system. Hardware cost and availability may make exact duplication of the original system unfeasible. It may be possible to perform initial tests and remediate on some systems using scaled down hardware. It should be noted that any difference between the original system and the test environment introduces a risk that the remediation will not be successful when moved to the original system. Care should be taken to minimize this risk as much as possible. It is highly recommended that the final

remediation of the code be tested as thoroughly as possibly on a hardware environment equivalent to the system it is to be implemented on.

As test cases are being developed, the need for test data to satisfy test case condition requirements will become clear. Test data can come from existing test cases or files. Data to support specific Year 2000 test cases can be created. Production data can be copied and used, but will not contain many high-risk dates or conditions. Adding specially developed test data to production data in order to satisfy high risk date testing requirements may help to overcome this shortcoming. Consider the use of date simulators to evaluate artificially changed calendar dates and the use of an appropriate tool to age data.

Create test data of dates on both sides of the end points of the valid date intervals, not just of the end points and the range within them. Testing invalid dates will provide an indication the system's response in such cases. Make sure all relevant "special dates are in test data and dates on either side of them as well.

5.2.5 Test execution

Test execution is the phase most likely to result in damage or loss to the system being tested.

As tests are being executed, special care should be taken to accurately identify and record those anomalies related to Year 2000.

The remediation technique may influence the selection of test tools to evaluate test results. For example, encoding may require a tool to interpret the codes used and report results.

Before executing a test case and between test case executions, it is necessary to assure the system is in the test base state. Restoring the system to the test base state after a test run may involve a number of actions including, in no specific order:

- Restoring the system time clocks, calendars
- Reconnecting any network or system
- Removing any test files and restoring the system to pre-test conditions
- Reinitializing (rebooting) the equipment
- Restoring the test data

5.2.6 Test verification and validation

5.2.6.1 Verification

Verification requires a correlation be achieved between the expected and observed results of a test case. Documentation should record either a correspondence between the two or make note of deviations. If failures occur that prohibit the completion of tests. The test cases should be documented along with the circumstances that precluded their use.

5.2.6.2 Validation

5.2.6.2.1 Test plan validation

Validation of the test plan attempts to ensure that the testing process is comprehensive.

As an example statistics gathered by tools that evaluate the number of branches or statements executed during testing versus the total number in the system's code are useful in showing the completeness of regression testing.

The techniques in **E.2 Coverage analysis** are useful in creating a valid test plan. If these methods or similar techniques were used during test planning their application should be documented during validation.

5.2.6.2.2 System validation

The validation process determines if the system is able to perform the function it was intended to perform. For the purposes of this document it is assumed that the system under test has been validated at some time. During the assessment and remediation process some functionality of the system may be abandoned in order to reduce the length of the Year 2000 process. System validation requires that checks be made to ensure that any changes to functionality have not affected the systems ability to perform its primary functions.

5.2.7 Summarizing and reporting

After the test cases that have been planned are executed, the test report is produced. Findings and recommendations are determined. Testing is complete when test completion criteria are met. Typical measures include tracking test cases completed to test objectives and/or measuring other coverage criteria.

Due to the potentially high business and safety risks associated with Year 2000 failures, it is important that test documentation be archived. See **Annex - C Example archive documentation (informative)**

5.2.8 Test reuse

In order to facilitate reuse of test cases, procedures and materials, they should be archived in such a manner as to recreate the test environment. The need to periodically re-create the test environment should be recognized and planned for deliberately. Year 2000 related testing is not a one-time event. Routine changes in system functionality will continue to occur and have to be tested. Regression (baseline) testing of all functions whenever change occurs is necessary due to the unanticipated effects of changes in one area upon other areas.

5.3 Impact of external constraints on the Year 2000 testing strategy

Technological constraints, business objectives, and the public welfare may be considered in designing an appropriate test strategy. The integration of a well-defined Year 2000 testing methodology will increase the likelihood of Year 2000 success as defined by the business goals of the project. Each system-in-test will have individual system elements that are at risk. Understanding the consequences of failure of the components can aid in prioritizing the focus of resources on individual testing tasks. Decisions as to whether any specific test methodology should be employed should be made against the background of these more general considerations. There are no universal solutions for the Year 2000 test problem. Factors to be balanced in deriving a test strategy may include but are not limited to:

5.3.1 Business/public considerations:

- Lost business
- Cost of failure
- Cost of litigation

- Regulatory Impact
- Public impact
- Public safety
- Shareholder value of public companies
- Fiduciary responsibilities of corporate or organizational officers.

5.3.2 Likelihood of failure

- Amount of changes
- Type of remediation
- Complexity of system elements
- Possibility of workarounds
- Interoperability of systems (i.e. contagion of failure)

5.3.3 Parallel development

Introducing non-Year 2000 related improvements along with Year 2000 remediation jeopardizes Year 2000 testing. This can invalidate the current Year 2000 test baseline making Year 2000 test results difficult to interpret.

5.3.3.1 Restrict new features

New features can introduce new requirements for system elements that have already been validated in the Year 2000 test process. This may require re-testing all or part of the elements previously validated.

5.3.3.2 Isolate elements

It may be necessary to temporarily isolate elements of a system. The isolated section of the system should be restricted from upgrades and bug fixes. Year 2000 remediation and testing can then be performed on the isolated component, while other elements remain available for normal development. As elements are validated, they are integrated into the development system and restrictions are lifted. Then a new area is isolated and restricted, and the process repeats. Since, development is restricted to areas not currently being remediated in the Year 2000 process; they can be integrated more easily into the Year 2000 system, thus limiting divergence of the systems.

5.3.3.3 Synchronizing remediation with ongoing development

The pervasive nature of some Year 2000 remediations can mean major structural system changes that requires months or even years to complete and test. During such an extended period of time, the system being revised for normal development may diverge so much from the Year 2000 remediated code that combining and re-testing becomes nearly impossible. Breaking the remediation and testing process into discrete components that can be completed quickly allows systems to be synchronized on a regular basis.

5.4 Test impact of remediation techniques

The relative effectiveness of each form of remediation depends on how centralized date handling is in the system element, the range of dates to be processed, and the time available to complete the project. The choice of remediation techniques has different testing impacts. If date handling is centralized, a relatively small number of sub-elements may need to be modified and tested. Drastically decentralized date handling, where data is accessed by separate code segments throughout the application, may require replacement.

5.4.1 Replacement (also called Migration):

Replacing a system element is likely to require a larger amount of testing to insure that all functionality, including Year 2000 specific conditions, operates as expected. Once a baseline for current operation has been established, transition tests and future regression tests should also be completed.

5.4.2 Windowing

Testing windowed remediation involves validating both the windowing algorithms and that all data falls within the window. Testing should assure that all dates that fall within a given date window are retained and dates outside that window are rejected.

The logic that allows the century of a 6-digit date to be interpreted may need to be implemented at any place where a date is processed. This means that the probability of remediation errors is very high because a large number of changes need to be made in complex areas of the system. All system elements that process windowed dates directly should be considered at risk.

Date windows restrict dates to a 100-year range. A system may have several different date windows that allow different data sets to have separate ranges. This can increase the complexity of testing by causing multiple boundary conditions to occur within a system or system element. Unit testing would require human interpretation of multiple date representations or the construction of reliable test rigs that could make the interpretations for the Test Engineer.

System elements that rely on windows should be protected against data corruption from input dates outside the window. If restrictions on entering dates outside the input range are enforced internally the element may be unit tested. Otherwise each data path that brings dates into the window should be checked separately.

5.4.2.1 Fixed

Fixed windows are established in the design of the system and are not accessible or changeable by the user. The testing impacts for this technique are similar to those discussed above. Assure that the correct window was chosen and that its date domain is adequate.

5.4.2.2 Movable

Movable windows are set by the user and are normally fixed at that point. The testing impact for this technique is similar to those discussed above. The user defined window (pivot year) means that the boundaries of the window are not known until the system is installed. The testing necessary for the data source and the window algorithm is complicated by the need to test multiple boundary conditions that represent the possible range of pivot years. It is possible that incompatibilities with other system elements may be apparent only with a specific set of boundaries.

5.4.2.3 Sliding

The sliding window technique bases its pivot year on the system date. Testing should ensure the system's ability to transition from one pivot year to another. Care should be taken to check that dates at one end of the window are not wrapped to another century when the pivot year changes. The other testing impacts for these techniques are similar to those for Movable Windowing Techniques.

5.4.3 Expansion

Expansion requires data and program modification. Testing the ability of system elements to interpret the new data type should be performed on every element that accesses the data directly. Changes in the input to an element may expand the range of its output beyond the original specification. It is necessary to insure that any required century information is included in the output, and that the element receiving the data can interpret it.

Expansion of permanent data stores usually requires the use of conversion programs. The conversion programs themselves need to be tested for accuracy in updating the entire range of dates in the current data store.

A larger date format may increase both the permanent storage and memory requirements of a system. Performance testing can be used to gauge what degradation may have occurred.

Expansion may require the creation of bridge programs to access existing data that, because of legal concerns, expense or for audit purposes, cannot be converted to the new date format.

Expansion normally requires modification of the size of dates in existing record formats. If data is accessed via an offset from a specific point in the record that offset may need to be changed to compensate for the larger date field. In some cases system elements which are not date sensitive but which happen to access the same data source may be at risk. This leads to greater levels of test effort.

5.4.4 Encoding

The encoding technique usually requires changing both data and logic. Like windowing, program logic may need to be tested and like expansion, other systems that access modified data should be tested to ensure their continued compatibility. Groups of applications that share date-data may be implemented simultaneously or implemented through temporary bridges.

The process used to encode existing data will normally include automated conversion utilities that should be tested.

Encoded data requires decoding for testing. This may add additional risk and resource requirements for the test procedure.

5.4.5 Elimination

The retirement of a system or application no longer deemed necessary.

5.4.6 Data encapsulation

It is possible to place bridges around existing data to allow their use within a system remediated to use a new date format. It is important to ensure that all access to this existing data is through the bridges. If other system elements access existing date data, expecting it to be in the remediated format, the remediated element may misinterpret the dates or suffer fatal errors. Data Encapsulation can eliminate the need for data conversion since conversion is accomplished on the fly. Instead of testing the conversion utilities or the converted data, the bridge is tested to insure its ability to correctly translate the range of possible data. It may be helpful to isolate the bridge system element(s) in order to effectively test. Since this method avoids the date transition, future date testing may be fully or partially eliminated.

5.4.7 System or system element encapsulation

There are several techniques that rely on bridges to isolate existing system elements from date-data outside the system elements working domain. For example bridges may be created that shift all dates back a preset amount of time allowing the system element to process some post 2000 dates as if they were actually prior to Year 2000 rollover.

Assuming that the system processes a range of dates prior to Year 2000 rollover properly, testing should ensure the bridges ability to shift dates in and out of that range accurately. Existing data may need to be modified since dates that were at the beginning of the systems old date range may be shifted out of range by the bridges. Error handling for out of range dates should also be checked.

Shifting dates can affect calculations that compute or are computed from the day-of-the-week, leap years, holidays, lunar phases, tidal cycles and other factors that occur on different dates from one year to the next. All dates that are shifted are at risk so tests should encompass the entire range of dates.

5.5 Special considerations involving embedded systems

Embedded systems pose numerous specific Year 2000 problems, many of which fall beyond the scope of this document. For example, the white box testing methods discussed involve analysis of programming logic. Embedded systems may be configurable by the user and may be interfaced with other systems or devices, but they may not be programmable by the user. Obtaining access to and interpreting the internal programming of embedded systems may be a complex and time-consuming process. Evaluation of risk and resource limitations may counsel against such efforts.

The black box testing methods discussed may often be more appropriate for embedded systems. Embedded systems may have specialized and circumscribed user interfaces, making it difficult to prepare and input appropriate test data to the system, and monitor and evaluate the associated system output. Embedded systems may function largely through specialized interfaces with other systems, for example, data acquisition and control and communications interfaces. Identification and correlation of the input and output of an embedded system interfaced within a larger, more complex system may not be an appropriate allocation of testing resources in light of the risks posed by a particular embedded system.

Nevertheless, many of the test procedures in clause **6 System elements at risk**, may be adapted for use with embedded systems. Examples include testing special-condition dates, date formats, event-triggers, and date interval and arithmetical calculations.

5.5.1 Supervisory distributed control systems (DCS)

Supervisory Distributed Control Systems contain Proportional Integral Derivative (PID) Control Centers that often operate asynchronous to the Supervisory DCS. PIDs do communicate with the Supervisory DCS. Alarms from the PIDs are prioritized as 1) Interrupt driven, for severe alarms, providing occurrence time-date stamps. 2) Polled Alarms for DCS recognition of critical and non-critical alarms, with process variable time-date stamps. Communicated data Logging, Trending, and Predictable functions with integral time-date stamps can also be event driven and aperiodic. The alarms are commonly provided to an alarm summary field and presented to plant operations as an alarm log summary screen, with a printout. Severe alarms are intended, in many instances, to shut down plant operations due to safety concerns. Should a corrupted data packet, time-date alarm occur, plant safety may be jeopardized.

PID Controllers and other asynchronous process control loops may contain independent Operating Systems, Firmware and Ladder Logic that runs independent of the Supervisory DCS Control. Independent control nodes support a given process and are self regulating, based on the process demands. Such loops contain system clocks that cumulate calendar functions based on arbitrary initial settings, input by field personnel. Seldom is time synchronization accurate or provided.

Event driven time-date inputs of alarms, and plant operation trigger points, are provided to the Supervisory DCS Control. During the Year 2000 Time Date transitions, the asynchronous and non-Year 2000 compliant time-date stamped data entering the Supervisory Control could cause fatal errors in programmed time-date driven functions, due to date misinterpretation.

5.5.2 User versus manufacturer testing

End users of embedded systems may find that their testing efforts are limited by a lack of adequate system configuration information, specifications, access to original code and proprietary testing tools, and availability of qualified testing personnel. For recently designed systems, the manufacturer may already have many of these resources in place. For this reason it is often more cost effective for the manufacturer to test the system. Users of older systems may find that the manufacturer will no longer support the system, making it necessary for the user to test the system or contract for testing with a qualified service provider. Even if Year 2000 certification is available from the manufacturer it may be prudent for the user to test the specific system in its current configuration. Reasons why a particular system or system element may fail while other apparently equivalent systems do not, include but are not limited to:

- Different components may be used in the assembly of the system tested by the manufacturer than are used in the system as installed. Although two integrated circuits or other components may be specified as functionally equivalent, difference in their implementation may cause one to have a Year 2000 failure while the other continues to function properly.
- Embedded systems are often elements of larger systems. As with other systems, improper interfacing between system elements can cause a failure. A manufacturer may not provide Year 2000 certification of an interface between two system elements either because there is a known incompatibility or because the interface has not been tested.
- Customization of embedded systems is often necessary to make them suitable to a specific installation. They range from minor modifications in control code to additional hardware requiring new functionality in the controller. A customized embedded system may be ordered from the manufacturer or the system could be modified during installation, maintenance, or system up-grades. Changes in hardware or code may cause the custom system to suffer functionality failures that do not occur on the standard system a manufacturer uses for testing.
- Documentation may specify methods for use, including limitations on date formats and ranges that could affect application of the technology in a broader systems context.

Interchange of data with other systems or components may require use of vendor defined "proper" formats.

5.5.3 System isolation to accommodate safety concerns

Embedded controls are often used in situations where there is a high risk to property or personnel. To reduce this risk, control systems can often be isolated from the systems they control. This allows the controller to be tested while virtually eliminating the physical

risks associated with the system. Generally this means disconnecting the controller and plugging its inputs into devices that can generate the signals needed for testing and its outputs into a monitoring circuit. Simulation techniques discussed elsewhere in this section are a good way to accomplish this and reduce the physical risk associated with testing these systems. In some cases it may also be possible to dry run a system or substitute less hazardous materials to reduce the risk of potential damage during testing.

5.5.4 Date simulators

The system time on some embedded systems may be difficult to change. This is particularly true in situations where critical functions are based on time date information or where synchronization is critical. In certain cases it may be possible to use a date simulator that either replaces the system's RTC or intercepts requests for the system date/time and returns an advanced system date. In either case it is necessary to ensure that all system date information is obtained from the simulation. If low-level functions are able to by-pass the simulation and access the system RTC directly, any tests run might be invalidated. It is also possible that multiple asynchronous RTCs might exist within the system. If one of these RTCs is not included in the date simulation the presence of two different system dates would, again invalidate testing.

5.5.5 Environment simulation

Simulating some hardware system elements can allow other elements to be tested in an off-line, controlled environment. In general, a simulation of a system element is designed from the functional specification of that element or by specifying and simulating its internal parts and building the simulation from these parts. A functional specification states how the system element is supposed to function, not how it actually performs. This means testing a simulation tests the specification, not the implementation. For this reason testing of a simulated system element should not be substituted for testing of the actual component. Simulations can be used to create a test environment for other system elements under test.

5.5.5.1 Logic testing

Simulating all hardware in an embedded system may allow initial Year 2000 testing of the system's code. This may be especially useful in situations where there is a possibility of damage to equipment or the environment, or injury to personnel. By isolating the code from the hardware it may be possible to find errors in the program's logic without physical risk. This type of simulation may also be used to allow acceptance tests of a code remediation to be performed off-line.

5.5.5.2 Bench testing

In certain situations it may be possible to remove a system element and test it as a unit. When testing embedded controllers, simple test scaffolding may not give sufficient information for the test engineer to recognize some types of errors. A simulation can be programmed to emulate the mechanical delays inherent in the system's hardware and provide sophisticated displays that make errors apparent to the tester. For example a controller attached to a robot arm that welds a part onto an assembly might have a delay in its program to allow the part to be positioned by another robot before the welding operation begins. Simple displays of the controller's output might show only the movements of the robot's joints. It might not be apparent if a failure in the delay caused the two arms to collide or caused the welding operation to complete before the part was in position. A simulation that allows the controller to operate a virtual model of the two arms might make this failure apparent without requiring an online test that might damage the robot hardware.

5.5.6 Asynchronous interfaces

Some embedded systems have multiple real time clocks that are set manually by operators and service technicians. The system design usually allows for these clocks to be slightly out of synchronization. An example would be, if day of the week information is not important to the computations based on one or all of these RTCs, then a six digit clock could rollover to 00-01-01 and continue to function properly, even without an indicator for the century. If the clocks are slightly out of synchronization and a comparison or calculation is made between date/times from two different clocks at Year 2000 rollover the difference between the date/times could appear to be nearly 100 years. These problems can be intermittent and difficult to track since the error may depend on which clock reaches rollover first and what process is running during transition.

5.5.7 Expansion statement

Expansion of date fields in embedded code may cause information adjacent to storage locations used for dates to be overwritten or corrupted. It is recommended that after expanding date fields the embedded code be functionally tested to ensure it still performs its intended function.

5.5.8 Test duration statement

As part of test planning it should be determined how long the subject device, component, or system should be running prior to establishing steady state conditions, how long it should run following a date change, and how long it should run to re-establish steady state conditions. Examples include:

5.5.8.1 Pretest

Steady state conditions should be established by running the device, component, or system for at least five minutes, or, if the device, component, or system performs a calculation function, then it should run for at least ten calculation cycles.

5.5.8.2 Test condition

Following a date change, the device, component, or system should be run long enough to determine the success, failure, or acceptability of the test. If the device, component, or system performs calculation functions, then it should be run for at least ten calculation cycles. Complex devices or components (those having multiple processors or sophisticated control software that performs error checking and corrections) should run for several hours as experience has shown that it may take an extended time for small errors to build to a critical enough fault for the device or component to fail. If no calculations are performed the run time following the date change should be based on the complexity of the device, component, or system. Simple devices and components should run for at least five minutes. Systems should also be run for several hours to allow for small errors to build to a critical enough fault for the system to fail. Each test plan should indicate the duration of the run time following a date change and provide some evaluation of its acceptability.

5.5.8.3 Power on power off test conditions

For power off/power on tests a sufficient time should be allowed to pass to ensure that any capacitor's have time to discharge. Physical inspection or vendor information may be used to determine if there are capacitors that need to be discharged.

Another consideration is devices or systems that have periodic update or data clean out functions. Tests for devices with these functions should be of sufficient duration for the device or system to go through the functions at least once. As an example, if a device does a data update every twelve hours, then the device needs to run for at least twelve

hours after the test data is input to ensure that the test data is read by the device and the appropriate pass/failure/acceptable deviation determination made.

5.5.9 Sampling statement

The purpose of sampling is to ensure that all devices or systems with the same model number have the same Year 2000 functional performance. Test plans should evaluate the need for a sampling plan when the test subject contains multiple instances of the same model devices or systems. One method for achieving this is to use knowledge or logic-based sampling using knowledge of chip sets and Operating Systems to group the devices or systems for testing. Alternatively, random sampling methods may also be applied when knowledge of chip sets and Operating Systems can not be obtained.

A useful example of knowledge or logic-based sampling is an organization which chooses to test a set of same model numbered devices with nearly consecutive serial numbers by selecting the earliest and latest serial numbered devices for testing for identical behavior. Given these devices exhibit identical behavior, a further decision might be made to sample randomly and test one or more of the devices within the serial number sequence. Given that all tested devices exhibit identical behavior the organization could then make the decision that all the devices in the original set exhibit the same behavior and results of the performed testing represents results for all the devices in the set.

5.6 Special-condition dates

The following is a list of dates that are recommended for consideration in the validation of Year 2000 remediation. This list is intended to be illustrative, but is not exhaustive. It may be appended or modified for a specific environment.

5.6.1 1900-01-01 (Monday)

The number of days in a century is not evenly divisible by 7 so no two consecutive centuries start on the same day of the week. If an algorithm disregards century information when making day of the week conversions, incorrect results may occur (See also 2000-01-01).

5.6.2 1999-01-01 (Friday)

In many systems the date is parsed into individual year, month and day variables and checks the validity of the date. Often an indicator is needed to trigger logic that reacts to a special situation. If the system insures that the variable is a valid date, a specific year value or date may be used to indicate the special situation. The year ‘99’ and dates within 1999 have been used for this purpose. When the reserved date occurs in normal data, the system may trigger special-condition logic that does not apply to the situation.

5.6.3 1999-09-01 (Wednesday)

The four digit date format (YY-MM) is sometimes used, with the three digit input 99-9 as a representation of an unknown or ‘out of range’ date. The input is interpreted to the full date 99-9-1 and stored. As long as the date is outside the range of normal data it is recognizable as ‘place-holder’ data. When the date 99-9 becomes a plausible entry for the field it becomes difficult to tell which 99-9 is a real date and which is a placeholder that needs to be replaced with real data.

5.6.4 1999-09-09 (Thursday)

This date is commonly used to indicate an unknown date in 6-character (i.e. 99-9-9) data entry fields that don’t require a leading zero. It was chosen because it was easy to type

and yet far enough in the future to be easily differentiated from 'real' dates. As 9-9-99 nears it will become impossible for the computer or the user to know if the entry is valid or not.

5.6.5 1999-09-10 (Friday)

In systems that have used 9-9-99 as a never expire date, logic allowing deletion of data after a specified date may fail to protect data that should be maintained forever.

5.6.6 1999-10-01 (Friday)

This is the first day of the US Government Fiscal Year FY2000.

5.6.7 1999-12-31 (Friday)

The last day that can be represented in standard 6-digit date format without Year 2000 rollover risk. This date is sometimes used to trigger special logic. It must be established that the system is able to distinguish between a regular end-of-year 1999 date and a special meaning date. For example, a license key intended to expire on 12-31-99 should not be confused with one that has no expiration date. This is also the start date for most Year 2000 rollover testing.

5.6.8 2000-01-01 (Saturday)

The first day of the Year 2000. There are many issues related to this date.

- A system with a day-of-week function based on 6-digit dates may change from Friday 1999-12-31 to Monday 2000-01-01 at Year 2000 rollover. 1900--01-01 was a Monday. A day of the week error could occur on any date after Year 2000 rollover if the calculation to derive the day of the week assumes that all dates have years between 1900 and 1999.
- There is a possibility that the date will be misinterpreted as 1900-01-01.
- Systems date counters may increment to erroneous dates like 19100-01-01.
- Parsing functions may misinterpret dates entered with one or both leading zeroes omitted. (See 5.6.12)

5.6.9 2000-01-03 (Monday)

This may be the first business day of the Year 2000. Certain business software calculates using proper business dates and days. Consequently, failures associated with this software would occur at this point. This day may be a holiday, not a business day. It is special from that perspective in that it would normally be a business day, but may not be in this case. It would be the first holiday of the Year 2000.

5.6.10 2000-01-04 (Tuesday)

This may be the first business day and first banking day in the Year 2000. Business applications may be sensitive to calculations using proper business dates and days. Consequently, failures associated with this software would occur at this point.

5.6.11 2000-01-07 (Friday)

This is the first Friday in the Year 2000.

5.6.12 2000-01-10 (Monday)

This is the first 7-digit date after Year 2000 rollover if leading zeros are not used for day and month representations. New or modified parsing functions required by changes in date-input formats or for interpreting an extended input range may fail when the number of digits representing the day changes. Parsing functions may need to be tested with all

possible combinations of dates with one- and two-digit month and year values and combination include and omit leading zeros. (Examples: 2000-1-1, 2000-01, 2000-01-1, 2000-01-01, 2000-1-10, 2000-01-10…) Each acceptable date representation should be checked to insure that it is correctly translated into the system's internal representation.

5.6.13 2000-01-17 (Monday - Martin Luther King day - USA holiday)

This may be the first Monday holiday in the Year 2000. This is a holiday that is always celebrated on a Monday rather than on a specific date in the USA. A day of the week calculation may be required to identify this date as a holiday. In other countries a similar situation may exist for locally celebrated holidays on other dates. These dates may need to be tested instead.

5.6.14 2000-01-31 (Monday)

This is the first month-end day in the Year 2000. Many programs key on month end as a trigger for a periodic function and may fail here.

5.6.15 2000-02-28 (Monday)

This date is not expected to cause any specific Year 2000 errors. Its relevance to testing is that it should be used as a start date in testing the system's ability to increment to 2000-02-29. This is necessary because it is possible for a system to recognize 2000-02-29 as a valid date when the date is manually set, while a separate system element increments the date directly from 2000-02-28 to 2000-03-01.

5.6.16 2000-02-29 (Tuesday)

The Year 2000 is a leap year. Program logic used to identify leap years may be incomplete. This could cause date processing errors for the remainder of the year.

5.6.17 2000-02-30 (Non-existent)

This day does not exist. Date functions should continue to recognize this as an invalid date.

5.6.18 2000-03-01 (Wednesday)

This is the first day after leap year day. The date calculations that transition from the last day of leap year February to the first day of March could fail. The possibility exists that some part of a system may fail to recognize Year 2000 as a leap year may lead to a condition where dates are no longer synchronized well as day of the week offsets can occur. This is also the first day of the last month in the first quarter of the Year 2000.

5.6.19 2000-03-31 (Friday)

This is the last day of the last month in the first quarter of the first year in the Year 2000. Quarter-end dates are significant in business and financial applications.

5.6.20 2000-04-03 (Monday)

This is the first business day of the second quarter of the first year in the Year 2000.

5.6.21 2000-04-17 (Monday)

Primary U.S. Income Tax due date in the Year 2000.

5.6.22 2000-04-28 (Friday)

April is the first business month whose last day coincides with a weekend in the Year 2000. This is the last business day of April.

5.6.23 2000-04-30 (Sunday)

This is the first month-end that coincides with a weekend in the Year 2000.

5.6.24 2000-06-30 (Friday)

This is the last day of the last month of the second quarter. This is the half-year point as well as the end of the fiscal year, or business cycle, for many businesses. This is the last business day of the first quarter end in the Year 2000 which coincides with a weekend.

5.6.25 2000-09-29 (Friday)

Last business day of the third quarter in the Year 2000.

5.6.26 2000-09-30 (Saturday)

This is the last day of the government fiscal year and last day of the third quarter of the Year 2000.

5.6.27 2000-10-01 (Sunday)

This is the first 7-digit date with a 2-digit month value. Parsing functions may need to be modified to allow for new date formats and a wider range of date-data during the remediation process. If leading zeros are not required for date-input, the date value entered might change the placement of numeric values for month, day and year within the input string. A parsing function that doesn't allow for this input variation might interpret 2000-10-1 as 0200-01-01 or might interpret the punctuation as part of the day or month value.

5.6.28 2000-10-10 (Tuesday)

This is the first date, after rollover that must be represented as an 8-digit date. Parsing functions may fail when the number of digits changes. This is similar to failures discussed in sec.5.6.27, but could result from misinterpretation of either the month or year value.

5.6.29 2000-12-31 (Sunday)

The last day of the Second Millennium of the Gregorian calendar. The ordinal date 1900-365 was the last day of 1900. Since 2000 is a leap year, its last day is 2000-366. An incomplete algorithm for determining the length of the year might cause an ordinal based system to transition into the new millennium a day too early.

5.6.30 2001-01-01 (Monday)

This is the first day of the third Millennium on the Gregorian Calendar. There is a possibility of errors in computing the day of the week. This is also a holiday and not a business day.

5.6.31 2001-01-05 (Friday)

This is the first Friday in the year 2001.

5.6.32 2004-02-29 (Sunday)

First leap day after Year 2000 rollover not affected by a century or millennium transition.

5.6.33 2004-12-31 (Friday)

This date can be used to determine if normal leap years are recognized by an ordinal date system.

5.7 Special date conditions.

5.7.1 Out of range

Any real date that can not be represented in the system's or system element's internal date format is considered an out of range date. Out of range dates might include dates outside a system element's date window, or before the start date of an integer offset date representation. The range of dates used will depend on the system implementation. They may be used in testing to determine the system's ability to protect data from corruption. If not recognized and rejected these dates may be wrapped into the acceptable range or misinterpreted in some other way, and introduced into the system. This could lead to data corruption.

5.7.2 Nonexistent dates

Dates with field errors, such as a month value of 13 or the 31st day of a 30-day month should be included in test data. This should be done to ensure that the system retains its ability to recognize these dates, reject them, and recover gracefully.

Common field error dates:

0000-xx-xx		2000-04-31	
1999-02-29		2000-06-31	
2000-xx-00		2000-09-31	
2000-00-xx		2000-11-31	
2000-02-30		2000-13-xx	

xx – Indicates a 'don't care'; any value in these fields indicates a nonexistent date

In some systems nonexistent dates may be used to indicate a special-condition such as an unknown or out of range date. If allowances are not made for the continued use of these dates the remediated system may reject or misinterpret them. This could result in a loss of system functionality. Tests for indicator dates should return a predictable result in accordance with the systems specifications.

Common indicator dates:

0000-00-00		2000-00-00	
1999-99-99			

5.7.3 Flag codes

Dates that have memorable patterns such as 99-9-9 or 99-1-1 may be used by some systems to override normal date processing and start a routine to handle specific condition flags.

5.7.4 Year 2000 rollover

The transition between 1999 and the Year 2000 may affect a date counter's ability to increment properly causing an error at the moment of transition. Comparisons and computations based on dates may give erroneous results if operations span rollover.

5.7.5 Leap year conditions

In the Gregorian calendar 1900 was not a leap year, but the Year 2000 is. A system or system element that represents both years as 00 may process both years as a leap year or as a non-leap Year. The ability of a system to recognize 2000-02-29 and not 1900-02-29 affects calendars and date displays as well as computations using a span of time including either of these dates.

5.7.6 Date comparisons using inequalities

It is often necessary to determine which of a pair of dates is prior to the other. If the date values being compared only include the two least significant digits of the year and they fall on opposite sides of the Year 2000 rollover, the operations used to compare them may return inaccurate results.
For example:

The 4-digit comparison gives the correct result,

2000-02-15 > 1999-12-15 = true

while its 2-digit equivalent gives an incorrect result

00-02-15 > 99-12-15 = false

5.7.7 28-year repetition

Any two dates between 1900-03-01 and 2100-02-28 that are 28 years apart will have the same day of the week associated with them. In cases where the results of tests are related to the day of the week, running the test cases twice with all dates 28 years apart allows the results to be compared for equivalence. This can help to eliminate the need to calculate test results by hand for these tests.

5.7.8 Computed day intervals

Time intervals may be computed by subtracting one date from another. If the operands for the subtractions are dates with only a 2-digit year, the computation may give incorrect results when the date range includes the Year 2000 rollover.

For example:

The 4-digit comparison gives the correct result,

2000-03-01 – 1999-12-31 = 61 days

while its 2-digit equivalent gives an incorrect result.

00-03-01 – 99-12-31 = -36464 days

5.7.9 Increment and decrement date by a count

Computing a target date a specific number of days prior to or after a given date may fail when the target and given dates would be on opposite sides of the Year 2000 rollover. Adding or subtracting enough days to cross the Year 2000 rollover may create a two-digit year value that is either negative or greater than or equal to 100. The system may not be able to convert these dates into standard date formats reliably.

5.7.10 Day of the week numbers

It is generally easier for a computer to manipulate numbers than strings so day of the week strings are often enumerated and referenced by the associated number. The day of the week number can be calculated by computing the interval between the date in question and a reference date and taking the result modulo 7. As noted above, the ability to compute date intervals is at risk, so the results of Day of the Week calculations based on them is also at risk.

5.7.11 Ordinal dates

When an ordinal date format is used, a leap year calculation based on the year value determines if the date XXXX-366 (where XXXX represents the year) exists in a particular year. Errors in identifying a year as leap year affect calculations of time intervals and conversions to other date formats.

5.7.12 Windowing dates

System elements that use date windows have defined boundary conditions that should be tested. Boundaries of windows may be different from the valid date interval of the system and should be specifically tested.

5.7.13 Day of the week

Calculation of the correct day of the week may involve logic to identify leap years, convert one or more date formats into Day of the Week numbers, interpret date windows or any number of other errors mentioned elsewhere in this list.

5.7.14 Calendar date to ordinal (Julian) conversion

Failure in these types of procedures is usually related to misinterpretation of leap year causing the resulting date to be off by one.

5.7.15 Local date formats

Systems that have multiple date display formats usually have an internal date representation that is converted to a formatted string for output. A separate function may be used to convert from the internal representation to each individual local date format. Errors can occur because the internal representation doesn't contain enough information to distinguish dates in Year 2000 from earlier dates. It is also possible that individual conversion functions may not be able to interpret the larger input values that represent Year 2000 dates.

The internal date representation of a system is usually set from an RTC using BIOS calls to transfer date-data from the RTC to the system clock. For globalization purposes a system may represent external date in a locally used calendar while one or all of these internal system elements may represent dates in the Gregorian system. Any one of the internal elements or the system element that converts the system date to a local date format may fail when the RTC encounters Year 2000 rollover. This can happen whether or not the local date format has any corresponding date transition.

5.7.16 Daylight savings time (Summer Time)

Daylight savings time changes occur on the first Sunday in April and the last Sunday in October in the United States. Other countries may have different dates for initiating and terminating the "Summer Time". This means that systems using time standards that include Daylight savings must use day of the week information to define if a date falls within the affected period. As described elsewhere in this list Day of the Week calculations may fail during Year 2000.

5.7.17 Year and month extraction values

The methods for extracting the day, month or year value of a date is dependent on the storage format of the date. Each format has its own potential problems.

5.7.18 Time zone offsets

Information passed between system and system elements is often associated with a time stamp. In Wide Area Network environments the receiving system may be in a different time zone than the originating system. This creates a situation where one system may rollover into Year 2000 while the other continues to operate in 1999 for several hours. If the time stamps include only 2 digit year values the receiving system may assume the information is invalid since the time stamp associated with it is either 100 years old or 100 years in the future.

In some cases, a system may use a single time reference, such as Coordinated Universal Time (UTC), ZULU time or Greenwich Mean Time, for all system elements regardless of their local time zone. This may be hidden from the user by converting date/time groups to local time for I/O purposes. A failure can occur when the offset between system time and local time crosses Year 2000 rollover.

It should also be recognized that not all time zones are an integral number of hours and conversion algorithms that use data that spans different time zones must be prepared to deal with these non-integral time zones. Examples of countries that contain such zones are parts of Australia, India, parts of Canada, Iran, Nepal, Sri Lanka and some Pacific Islands. Further, some Pacific Islands have a shift to Daylight Savings Time that is not a full hour and that also must be taken into consideration if the data is from or is intended to be transmitted to such places.

5.8 System components to be tested

Effective system testing may involve the creation of test cases that act on or require various combinations of the following system components. The test cases presented in the following sub-clauses should be considered in the context of each component.

5.8.1 System element

System elements may be tested before and after remediation. This can include but is not limited to software, hardware, and firmware. Test cases should validate that no desired pre-existing functionality has been lost (regression testing) and that new 'Year 2000' functionality works as expected (compliance testing).

5.8.2 System data

Systems should be tested using data having current dates and dates that have been advanced to beyond the Year 2000. Date-data that have been advanced beyond the Year 2000 may be referred to as aged data. Existing data can be aged or Year 2000 data set can be created to meet testing requirements. Systems should also be tested with the system date in or post Year 2000 with data from before year 2000.

5.8.3 System time

Systems may be tested using current system time and with the time set beyond the Year 2000. Setting a system time to beyond 2000 may allow the testing of many system factors not normally observable using current time. These factors include the operating system utilities that use internal time stamps, system archiving or backup utilities and other aspects of the internal operation of a system.

5.8.4 Data and time combinations

Test cases should represent all combinations of system time and date-data that are possible during the system's transition into the Year 2000.

- System time prior to Year 2000 with data before Year 2000.
- System time prior to Year 2000 with data after Year 2000.
- System time post Year 2000 with data after Year 2000.

System time post Year 2000 with data before Year 2000.

6 System elements at risk

This section is an attempt to outline example system elements or features that have a potential of being affected by date based algorithms and, therefore, could fail if not designed to cross the Year 2000 rollover. Special condition dates and example test

assertions are included that are generic in nature and correlate with certain system elements and that can be customized for specific needs.

6.1 Interfaces: Shared control blocks/API/DDE/OLE

6.1.1 Definition

(API) Application Program Interface -The interface by which an application program accesses operating system and other services. An API is defined at source code level and provides a level of abstraction between the application and the implementation.
(DDE) Dynamic Data Exchange - A protocol that allows application programs to communicate using a client-server model. Whenever the server modifies part of a document, which is being shared via DDE, one or more clients are informed and include the modification in the copy of the data on which they are working.
(OLE) Object Linking and Embedding - Allows an editor to place part of a document into another editor and then re-import it.

6.1.2 Rationale

In transferring date information, these functions may use an intermediate date-data form, which might not support proper handling of century date information. It is necessary to compare source information to destination information to ensure accurate date-data transfer.

6.1.3 Related elements - none

6.1.4 Test cases

6.1.4.1 OLE

Change of a date field in an application results in a corresponding change in a linked application.

I Test objective

Verify that changing a linked high risk Year 2000 date that is in a Master application (e.g. spreadsheet, database and presentation) will change the corresponding date in a linked Slave application.

II Test conditions

Two applications with linked date-data. Have a master application one with date field that is linked to a corresponding field in a slave application.

III Procedure/results

#	Test procedure	Expected test results
1	Master – Change date field in application to one of the high-risk Year 2000 dates.	
2	Slave - View the date field in second application that has an OLE link.	The date-data field will be updated to high-risk Year 2000 date.

6.2 Archiving/restoring

6.2.1 Definition

The transfer of data files to and from a long-term storage device, for example magnetic tape.

6.2.2 Rationale

Most archives specify a retention date or interval. Archiving systems must be able to accurately compute and compare these dates through Year 2000 rollover and beyond 2001. If the dates used don't include century information an archive recorded in '97 with a retention interval of 5 years would be dated for deletion in 02 and might be considered '95 years old at its creation.

The problem is compounded by the use of certain dates as special logic indicators. In some applications year fields with the values 00 or 99 are used as indicators for records that have no expiration. Without modification, these applications might save all archives dated for deletion in 1999, 2000 or on specific days in 1999 and 2000 indefinitely. Depending on the retention duration, some systems may already be storing archives that should be deleted when their retention date has passed along with archives that use the same date value to indicate that they should never be deleted. Leaving no easy way for the system to distinguish between them.

It is also necessary to insure that the remediated system can restore archives that existed before remediation. This may be accomplished by converting legacy archives to the remediated date format as part of the remediation process, by giving the restore system element the capability to convert Legacy data to the remediated format on the fly, or by encapsulating the legacy data archives. It is highly recommended that a copy of the unmodified archive system be maintained to perform restorations in case there is a compatibility issue with legacy archives.

6.2.3 Related elements

Backup & Restore
Calculations

6.2.4 Test cases

6.2.4.1 Archiving/restoring files

6.2.4.1.1 Deletion on specified date

Some archives allow the user to assign an archive expiration date and tag the archive for automatic deletion. This may be used as a security feature to keep unnecessary sensitive information from accumulating on systems. It is necessary to verify that the system will not prematurely delete needed information while retaining its ability to purge those files that have passed their expiration dates. The specification for the example system states that the archive is retained until the last minute of its expiration date.

I Test objective

Verify that an archive can be set to automatically purge on a specified date during the Year 2000.

II Test conditions

#	**Conditions**
1	Standard operating environment

2	Set of test files to be archived

III Procedure/results

#	Test procedure	Expected test results
1	Set the system date to a date prior to 2000-01-01 Year 2000 rollover. Create an archive. Set the archive retain date to 2000-01-01. Set system date and time to 2000-01-01 23:50:00. Open the archive and view its contents.	Archive should open and have all archived files.
2	Close the archive and wait until system clock reads 2000-01-02 00:01:00. Attempt to open the archive.	A message should say that the archive does not exist.
3	Repeat steps 1 and 2 with the following dates. 2000-01-10 2000-02-29 2000-10-01 2000-12-31	The above results 1 and 2 should be repeated for each date.

6.2.4.1.2 Deletion after retention period

Automatic archive systems may periodically delete older archive files. This function may be used in situations where documents have to be held for a specific period in order to meet legal requirements. The sample test case is written for a system that requires the archive be saved until midnight of the last day of the retention period. This could be specified as:

IF ((Archive_Creation_Date + #Days_To_Retain) < Current_Date) Delete Archive

Note: The comparison requires the number of days in the retention period. The system may have to calculate this value from month or year values or combinations of month, day and year (i.e. 3 months and 11days). See sub-clause **6.4 Calculations**.

I Test objective

Verify that archive files will be purged after a specified interval that spans Year 2000 rollover.

II Test conditions

#	Conditions
1	Standard operating environment
2	Set of test files to be archived

III Procedure/results

#	Test procedure	Expected test results
1	Set the system date and time to 1999-12-10 12:00:00. Create an archive with a retain time of 1 year. Set system date and time to 2000-12-10 23:50:00. Open the archive and view its contents.	Archive should open and have all contents specified at its creation.
2	Close the archive and wait until	A message should say that the

	system date changes to 2000-12-11. Open each archive.	archive does not exist.
3	Repeat steps 1 and 2 for the following dates and retain times.	The above results 1 and 2 should be repeated for each set of inputs.

First system date	Retain time	Second System Date
1999-12-10	31 days	2000-01-10
1999-12-01	3 months	2000-02-29
1999-12-01	11 months	2000-09-31

6.2.4.2 Date expirations set at 1999

Some tape systems use a 2-character year with 3 characters representing the day of the year to encode expiration dates. The date 1997-02-01 is stored as 97032 in this representation. When it is necessary to create archives that have no expiration, the dates 99365 or 99366 may be used as indicators. Since many archives are already marked with these dates, a remediation of this type of system should allow for their continued use. This could cause confusion, as program logic will need to distinguish between tapes intended to expire at the end of 1999 and those that use that date to indicate they can never expire.

6.2.4.2.1 Date expirations - Test Case One

I Test objective

Verify that archive tapes with a retention date of 1999-12-30 or 1999-12-31 will expire at expected time.

II Test conditions

#	Conditions
1	System date set to a date early in 1999
2	A set of files to archive
3	Archive application set to overwrite only if the archive retention date has expired.

III Procedure/ results

#	Test procedure	Expected test results
1	Set archive retention date to 1999-12-30. Create an archive. Set the system date to 1999-12-30 at 23:55:00. Attempt to overwrite the archive.	A warning should appear stating that the archive is not expired and the archive should be retained.
2	Wait for the system to reach midnight and move into the next day. Attempt to overwrite the archive.	The system should allow the archive to be replaced
3	Set the system back to a date early in 1999. Repeat steps 1 and 2 with the date 1999-12-31.	The results 1 and 2 should be repeated.

6.2.4.2.2 Date expirations - test case two

I Test objective

Verify that existing archives that are marked as, “Never-Expire” won’t be

interpreted as expired after Year 2000 rollover.

II Test conditions

#	Conditions
1	Scratch archive tape created on the modified system with the never-expire indicator set.
2	Archive application set to overwrite only if the archive retention date has expired.
3	Test system with remediated archive system installed.

III Procedure/ results

#	Test procedure	Expected test results
1	Set system date to 1999-12-31 at 23:55:00. Attempt to overwrite the archive.	A warning should appear stating that the archive is not expired and the archive should be retained.
2	Wait for the system to reach midnight and move into the next day. Attempt to overwrite the archive.	A warning should appear stating that the archive is not expired and the archive should be retained.

6.2.4.3 Archive sequence

Archiving systems may create directories or views of the contents of an archive that are ordered by the modification date of individual files in the archive.

I Test objective

Verify that the files in an archive can be viewed in order by date when files were created after Year 2000 rollover.

II Test conditions

#	Conditions
1	Change the system date to 1999-12-15 in YYYY-MM-DD format. Create a new file in the application.

III Procedure/ results

#	Test procedure	Expected test results
1	Set the system date to 1999-12-31. Create a file.	The file date should read 1999-12-31
2	Repeat step 1 for system date 2000-01-01.	The file date should read 2000-01-01.
3	Repeat step 1 for system date to 2000-2-28.	The file date should read 2000-2-28.
4	Repeat step 1 for system date 2000-02-29.	The file date should read 2000-02-29.
5	Repeat step 1 for system date 2000-03-01	The file date should read 2000-03-01.
6	Create an archive containing the files created in steps 1-5.	The archive should be created without error.
7	View the archive in increasing ordered by date.	1999-12-31 should be at the top of the list 2000-03-01 should be last.
8	View the archive in decreasing ordered by date.	2000-03-01 should be at the top of the list 1999-12-31 should be last.

6.2.4.4 Preexisting archive

As mentioned in the rational for this section, preexisting archives may contain dates in a different date format than the remediated system uses. At some point during the restoration process it may be necessary to convert data from the old date format to the post-remediation date format. This test case checks for data corruption in the conversion process. The test is written using the concept that two separate systems are maintained. It might be more cost effective in some cases to create the legacy test archives on the unremediated system then convert that system to the remediated configuration before completing the test.

I Test objective

Verify that the dates in a preexisting archive are converted to equivalent dates in the remediated date format when they are restored.

II Test conditions

#	Conditions
1	System A: An unremediated system capable of creating an archive in the legacy format.
2	System B: A system that is a Year 2000 remediated version of system A.

III Procedure/ results

#	Test procedure	Expected test results
1	Create an archive on system A with a series of data that includes the boundaries of the legacy systems valid date interval.	Archive should be created without error
2	If the remediated system requires that legacy archives be converted prior to restoration perform the conversion.	The conversion process should complete without error
3	Restore the archive on system B and view the restored data.	Dates from the restored archive should be equivalent to the data on system A represented in System B's date format.

6.2.4.5 Manual deletion

The sample test case is written for a system that computes the retention date of the archive by adding a user-defined retention period to the current system date. An attempt by a user to delete an archive that has a retention-date after the current system date should produce an error or warning message. In some systems special access or privileges may be required to perform these deletions. This test case assumes that the system does not allow the deletion of an archive before its expiration. It is also possible that a system using a 2-digit year in an expiration date might not allow the deletion of an expired archive after Year 2000 rollover. For example, an archive with a 6-digit retention date of 99-12-01 might be considered current when the system date is 2000-01-01 if the comparison is made base on a 2-digit year.

I Test objective

Verify that a system allows manual deletion of an archive after the system time passes its retention date but not before.

II Test conditions

#	Conditions
1	A system with an archive function that allows a user to set the

	archive retention period.

III Procedure/results

#	Test procedure	Expected test results
1	Set System date to 1998-06-06	
2	Create archive A and B with 1-year data retention	
3	Create Archive C with 2-year data retention	
4	Set system date to 1998-12-01	
5	Attempt to delete archive A and B	Should get an error message stating that archive should be retained until 1999-06-06
6	Attempt to delete archive C	Should get an error message stating that archive should be retained until 2000-06-06
7	Set system date to 1999-12-01	
	Attempt to delete archive A	Archive A should be deleted normally
	Attempt to delete archive C	Should get an error message stating that archive should be retained until 2000-06-06
	Set system date to 2000-01-01	
	Attempt to delete archive B	Archive B should be deleted normally
	Attempt to delete archive C	Should get an error message stating that archive should be retained until 2000-06-06
	Set system date to 2000-06-07	
	Attempt to delete archive C	Archive C should be deleted normally

6.3 Backup and restore

6.3.1 Definition

To make a copy of a file, file system or other resource, that will be replaced in the event of failure or loss of the original.

6.3.1.1 Full backup

The creation of a backup of all files on a particular storage device or system.

6.3.1.2 Partial backup

The creation of a backup of a subset of files selected by the user.

6.3.1.3 Incremental backup

The creation of a partial back up consisting of those files which have been modified since they were last backed up.

6.3.2 Rationale

The scheduling function of a backup routine must continue to operate properly during the transition to the Year 2000 and beyond. The triggers that start these automated functions

should be checked for proper operation. The conditions that select a file for restoration or backup may be based on file dates. The ability of the routine to correctly interpret and compare file dates to backup dates should be tested.

6.3.3 Related elements

Event triggers
Archiving
Error handling

6.3.4 Test cases

6.3.4.1 Full backup/overwrites

The backup system may restrict the user's ability to overwrite backup files with data that is older than the backup file data. This system could interpret Year 2000 files as older than files created before 2000 and not allow the backup to continue.

I Test objective

Verify that backups made prior to the Year 2000 can be successfully overwritten by backups made after the Year 2000.

II Test conditions

#	**Conditions**
1	Set the system clock to 1999-06-06.
2	Test data exists with dates ranging from the beginning of the test period to 1999-06-06. This test data may constitute a set of files within a directory, an entire drive, a database or other data storage structure.

III Procedure/ results

#	**Test procedure**	**Expected test results**
1	Create a full backup of test data.	A backup is created with the date 1999-06-06.
2	Change the system date to 2000-03-01 and modify the test data.	
3	Overwrite original full backup and delete test data on source system.	A backup is created with the date 2000-03-01and all test data on the source system is removed.
4	Restore backup from step 3 and check test data for the modification.	Content is the same as the original test data with the modifications made in step 2.

6.3.4.2 Selection of files

Logical operators used for the selection of files may need to compare dates to determine what files to include in the backup. Improper implementation could cause a large amount of unwanted information to be included, while the intended data is left out. The basic issues are:

- When attempting to backup data created after 2000-01-01 or a later date, the system includes some data that is dated in the 1900's.

  2000-01-01 < 1998-02-15 = False ⇒ File is excluded [1]

  00-01-01 < 98-02-15 = True ⇒ File is included

- When attempting to backup data created prior to 2000-01-01 or a later date, the system does not include some data that is dated in the 1900's.

  2000-01-01 > 1998-02-15 = True ⇒ File is included

  00-01-01 > 98-02-15 = False ⇒ File is excluded

- When attempting to backup data created after 1999-12-31 or an earlier date, the system does not include data that is dated in the 2000's.

  1999-12-31 < 2000-01-01 = True ⇒ File is included

  99-12-31 < 00-01-01 = False ⇒ File is excluded

- When attempting to backup data created prior to 1999-12-31 or an earlier date, the system includes some data in the 2000's.

  1999-12-31 > 2000-01-01 = False ⇒ File is excluded

  99-12-31 > 00-01-01 = True ⇒ File is included

The basic tests should include cases to test each of these issues. The example given tests only the greater-than-or-equal-to operator. Similar tests should be developed to test all other inequalities used in this type of comparison.

I Test objective

Verify that the backup procedure will correctly select and backup files created after a specified date.

II Test conditions

#	**Conditions**
1	Create files with the following creation/modification dates
2	1998-01-01, 1999-12-31, 2000-01-01, 2001-12-31, 2004-12-31

III Procedure/ results

#	**Test procedure**	**Expected test results**
1	Create a backup of files created on or after 1999-06-06 and check its contents.	Backup 1999-06-06 should contain files 1999-12-31

[1] The symbol '⇒' is used as a logical operator for exclusive implication. It indicates that if and only if the left side of the expression is true the right side must also be true for the equation to evaluate as true. In each pair of equations on this page the first equation represents the correct evaluation of the equation using a 4-digit year. The second equation in each pair shows how a system might evaluate the left side of the implication differently when processing dates with 2-digit years. The right side of the implication shows the resulting change in functionality for the system.

		2000-01-01 2001-12-31 2004-12-31
2	Create a backup of files created on or before 2001-06-06 and check its contents.	Backup 2001-06-06 should contain files 1998-01-01 1999-12-31 2000-01-01

6.3.4.3 Automatic

The next scheduled backup or last backup date may be stored by the system. It is then compared to the current date at regular intervals to determine when the next backup should occur. Errors in the system's interpretation or comparison of either of these dates may cause the backup process to fail. The sample test case is for weekly backups. Since different logic is often used to interpret intervals such as months, quarters or years, each interval used should be tested separately.

I Test objective

Verify that automatic data file backups will occur on schedule across Year 2000 rollover.

II Test conditions

#	**Conditions**
1	Set backup parameters to Full Backup.
2	Set backup frequency to weekly. Set backup date/time to Friday - 00:05:00
3	Set of test data files.

III Procedure/ results

#	**Test procedure**	**Expected test results**
1	Set the system date to 2 min. before backup is scheduled to occur, i.e., 1999-12-31T00:03:00. Wait for backup to complete.	Backup with date 1999-12-31 is created.
2	Create a new test data file on source system.	The file has been created.
3	Set the system date/time to 2 min. before next backup is scheduled to occur, i.e., 2000-01-07T00:03:00. Wait for backup to complete.	Backup with date 2000-01-07 is created. Verify that file created in step 2 is contained in the backup.

6.4 Calculations

6.4.1 Definition

The performance of arithmetic functions on date/time information.

6.4.2 Rationale

Systems or system elements may perform calculations using date-data. During the Year 2000 rollover, there is a risk that date calculations may produce incorrect results in a

variety of ways. For example, it is often necessary to calculate the interval between two dates. Calculation of intervals with 2-digit dates can yield invalid results:

2001 - 1997 = 4
01 - 97 = -96

Some systems or system elements may only accommodate two digits for year-data input or may only perform calculations using two digits of a four-digit year. Errors may occur in the following types of calculations:

- Calculation of task duration
- Calculation of Interest payments
- Payroll and holiday calculations
- Leap year calculations

6.4.3 Related elements

Archival and retrieval

6.4.4 Test cases

6.4.4.1 Calculation of task duration

This example refers to a project planning application in which a task is a series of actions that require a nontrivial amount of time defined by a start date, an end date and a duration. These values are then used in creating schedules, estimating costs and assigning personnel. When given the start date and either the end date or the duration the application must be able to calculate the unknown variable.

6.4.4.1.1 Across Year 2000 rollover

I Test objective

Verify that the duration of a new task across the Year 2000 rollover is correctly calculated.

II Test conditions

#	**Conditions**
1	Change the system date to 1999-12-31
2	Set the system for a 7-day workweek.

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Create a task in scheduling application/software with a start date of 1999-12-31 and an end date of 2000-01-04. Check reported duration of task	Task should show a duration of 5 days
2	Create a task with a start date of 1999-12-31 and duration of 6 days. Check task end date.	Task has an end date of 2000-01-05
3	Create a task with an end date of 2000-01-05 and duration of 6 days. Check task start date	Task start date should be 1999-12-31

6.4.4.1.2 Calculation of Interest

The following test cases assume annually compounded interest over a one year period. This is the least complex case. It is used here for clarity. Other cases may be necessary for interest compounded quarterly, monthly, daily or for other cycles used by a particular system.

6.4.4.1.2.1 loans

I Test objective

Verify that interest calculations are accurately performed over the Year 2000 rollover.

II Test conditions

#	Conditions
1	A computer system with interest calculation software.
2	Date of system is set to 1999-06-01

III Procedure/results

#	Test procedure	Expected test results
1	Generate a fictitious $10,000 Loan for a one year duration. Set interest rate for 10% compounded annually	
2	Advance system date to 2000-06-01	
3	Check interest due	Interest payment should be $1000.00

6.4.4.1.2.2 Savings account.

I Test objective

Verify that interest calculations are accurately performed over the Year 2000

II Test conditions

#	Conditions
1	A computer system with interest calculation software.
2	Date of system is set to 1999-06-01

III Procedure/results

#	Test procedure	Expected test results
1	Open a fictitious savings account with $10,000 in it. Set interest rate for 10% compounded annually.	
2	Advance system time to 2000-06-01.	
3	Check accrued interest.	Interest accrued should be $1000.00

6.4.4.1.3 Across leap year in 2000

Some date calculation functions indicate a leap year and thus insert an additional day in February, February 29th.

This test method attempts to verify Leap year calculations as accurate. February 29, 2000 is a valid date by virtue of the fact that the Year 2000 is a leap year. This is especially important since the Year 2000 may not be recognized as a leap year by some incomplete implementations of leap-year algorithms. It should be noted that 1900 and 2100 are NOT leap years. If an application requires those dates, then this test method may be modified to suit.

I Test objective:

Verify a system or system element's ability to include a leap day in the Year 2000 and not in 2001. Note that the test focuses on a scheduling application; the actual implementation of the test may be modified to fit any number of applications.

II Test conditions:

#	**Conditions**
1	Change the system date on the business server and on the remote client to 2000-02-28.
2	Set the system for a 7-day workweek.

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Create a task with a start date of 2000-02-28 and an end date of 2000-03-03. Check the task's duration.	Task should show duration of 5 days.
2	Create another task with a start date of 2000-02-28 and duration of 5 days. Check the task's end date.	Task has an end date of 2000-03-03.
3	Create another task with an end date of 2000-03-03 and duration of 5 days. Check the task's start date.	Task has a start date of 2000-02-28.
4	Create a task with a start date 2100-02-28 and end date 2100-03-03. Check the task's duration.	Task should show duration of 4 days.
5	Create a task with a start date of 2100-02-28 and duration of 5 days. Check the task's end date.	Task has an end date of 2100-03-04.
6	Create a task with an end date of 2100-03-03 and duration of 5 days. Check the task's start date.	Task has a start date of 2100-02-27.

6.4.4.2 Holidays within a Leap Year

If a leap year is not appropriately recognized, secondary failures may occur, such as erroneous recognition of holidays. Certain systems or system elements may need to identify holidays for various purposes (e.g., payroll calculations). Holidays may have a fixed value, such as the US Independence Day holiday, July 4. Other holidays may be derived such as the US holiday Thanksgiving Day, which occurs on the fourth Thursday of November. These tests attempt to verify that holidays occurring in a leap year are appropriately recognized.

I Test objective

Verify that an application calculates correct pay for hours entered on holidays in a leap year.

II Test conditions

#	**Conditions**
1	Base pay is $10.00 per hour
2	Work during holidays is considered double time.
3	Thanksgiving day and Independence day (US), 2000-11-23 and 2000-07-04, are considered holidays. 2000-07-03 and 2000-11-22 are NOT holidays and are compensated at straight time.

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Create a pay period for an employee starting on 2000-07-02 through 2000-07-08. Enter two hours worked on the 3rd and no other time worked in that pay period.	Pay should be $20.00
2	Create a pay period for an employee starting on 2000-07-02 through 2000-07-08. Enter two hours worked on the 4th and no other time worked in that pay period.	Pay should be $40.00
3	Create a pay period for an employee starting on 2000-11-19 through 2000-11-25. Enter two hours worked on the 22nd and no other time worked in that pay period.	Pay should be $20.00
4	Create a pay period for an employee starting on 2000-11-19 through 2000-11-25. Enter two hours worked on the 23rd (Thanksgiving day) and no other time worked in that pay period.	Pay should be $40.00

6.5 Date determination

6.5.1 Definition

Functions that return date information or change the value of date/time sources.

6.5.2 Rationale

These system elements are the source, either directly or indirectly, of all date information in a system that has not been input by a user. At the lowest level they must directly interpret hardware counters converting a simple binary string into character data in a variety of formats. As the Year 2000 approaches, limitations that were too far in the future to concern the original programmers may become important issues. An example might be a date setting function that accepts only 6 characters of input making it impossible to set the system to a date after 1999-12-31. More complex functions use the

outputs of lower-level functions to calculate days of the week, the occurrence of holidays or if the current year is a leap year. Some date sources may cease to function during Year 2000 rollover or they may start returning dates in the early 1900's.

6.5.3 Related elements

(API) Application Program Interface

6.5.4 Test cases

6.5.4.1 Weekend recognition for overtime/day of the week

This test case uses a difference in pay rates between weekdays and weekends as an indicator that the system correctly identifies weekend versus non-weekend dates. The week 2000-06-04 – 2000-06-10 was chosen because it has no holidays to confuse the issue. The hours entered for all days should be small enough that overtime rules for number of hours per day and total hours per day do not have an affect.

I Test objective

Verify that application will assign overtime to the payroll reports for hours of work done on weekends after the Year 2000.

II Test conditions

#	**Conditions**
1	System date set to 2000-06-12
2	Payroll program running
3	A time period for the week 2000-06-04 – 2000-06-10
4	At least one task to assign work to.
5	At least one full time resource with standard pay rate $10.00 (weekends paid as 1.5 time's hourly wage).

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Enter the following hours worked for the resource on the indicated days. Check the payroll report. Sunday - 4 hours Monday – 8 hours Tuesday – 8 hours Wednesday - 8 hours Thursday – 8 hours Friday - 0 hours Saturday - 1 hour	Total time for the resource is 37 hours. 32 hours standard time and 5 hours overtime. Total weeks wage $395.00. 395= (32*10) + (5*15)

6.5.4.2 Automated display functions of calendars

By setting the system date prior to January 2000 it is insured that the logic needed to define dates as holidays can function across Year 2000 rollover. Results are given for U.S. holidays and events and will be different in other countries. The years 2000 and 2001 are used as examples of a Leap Year versus a Non-Leap Year.

I Test objective

Verify that the correct days are annotated as holidays on the calendar display in the Year 2000 and beyond.

II Test conditions

#	**Conditions**
1	Set system time to a date prior to 2000-01-01
2	Set calendar to display all US holidays

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Change calendar year to 2000. Check that holidays are displayed on the dates specified in the table.	The dates in the 2000 column of the results table should be listed as holidays
2	Repeat step one for the year 2001. (Testing may continue beyond 2001 if holiday dates are from a reliable source.)	The dates in the 2001 column of the results table should be listed as holidays

IV Results table

Holiday or Event	**2000**	**2001**
New Years Day	01-01	01-01
Martin Luther King Day	01-17	01-15
Lincoln's Birthday	02-12	02-12
Valentine's Day	02-14	02-14
Presidents Day	02-21	02-19
St. Patrick's day	03-17	03-17
Easter	04-24	04-15
Mother's Day	05-14	05-13
Memorial Day	05-29	05-28
Flag Day	06-14	06-14
Father's Day	06-18	06-17
Independence Day	07-04	07-04
Labor Day	09-04	09-03
Columbus Day	10-09	10-08
Halloween	10-31	10-31
Election Day	11-07	11-06
Veterans Day	11-11	11-11
Thanksgiving	11-23	11-22
Christmas	12-25	12-25

6.5.4.3 Holidays time periods/ automated functions

The effects of days of the week, holidays and leap years may not be correctly interpreted when starting automated functions after the Year 2000. The following test case is an example that requires correct interpretation of a holiday. It involves a bank security system that will not allow access to the vault on weekends or bank holidays. The holiday used occurs prior to February 29th to isolate the test from the effects of an incorrect interpretation of leap year. Similar test cases include tests for the proper interpretation of normal weekends, fixed-date holidays, and variable-date holidays both before and after February 29th in Leap Years and Non-Leap Years.

I Test objective

Verify that vault security systems require special access to open on Martin Luther King day in the Year 2000

II Test conditions

#	Simulation of vault door accesses security.
1	Set weekend access to start Fridays at 18:00:00 and end Monday at 07:00:00.
2	Set all holidays to no access.

III Procedure/results

#	Test procedure	Expected test results
1	Set the simulation time to 2000-01-14 - 17:55:00. to open vault using normal week day procedures	Access should be granted normally.
2	Close vault and wait 10 minutes. Attempt to open vault.	Access should be denied.
3	Advance system time to 2000-01-17 - 07:05:00 Attempt to open vault.	Access should be denied.
4	Advance system time to 2000-01-18 - 06:55:00 Attempt to open vault	Access should be denied.
5	Wait 10 minutes. Attempt to open vault.	Access should be granted normally.

6.5.4.4 Dates that are calculated by mathematical operation

Date calculations are often either additions or subtractions of a time interval. In this case the interval is calculated by a formula that requires day of the week information. This scheduling function must determine the number of hours that each assigned worker has available on the start day of the task and subtract that value from the total hours required to complete the task. If the task is still not complete it repeats the process for the following day. It then adds the count of the number of days required to the start date. Failures can be caused by retrieval of the wrong availability or a mistake in adding the interval to the start date.

I Test objective

Verify that task completion dates may be marked using the estimated worker-hours, number of assigned personnel and the start date.

II Test conditions

#	Conditions
1	Open a test project with a starting date of 1999-12-31 and at least two workers that have no currently assigned tasks.
2	Set both workers schedule to 8 hours per day weekends and holidays off.
3	2000-01-01 and 2000-01-02 fall on a weekend. For purposes of this test case 2000-01-03 is treated as holiday in lieu of New Years day.

III Procedure/results

#	Test procedure	Expected test results
1	Create a task with an estimated duration of 30 hours. Do not include a completion date.	Task should be indicated as unassigned.
2	Open a project schedule. Assign the task to both employees on the first available date. Assign the	

	task to employee 1 alone on subsequent days until it is completed.	
3	Check task completion date.	Task completion date should be 2000-01-06

6.6 Hardware

6.6.1 Definition

For purposes of this test method, hardware, as distinguished from firmware or software, can be considered that physical component within a system or system element that is responsible for tracking and maintaining time and date. An example of hardware is the integrated circuit within a PC that performs the function of *Real Time Clock*.

6.6.2 Rationale

Many systems or system elements use a hardware sub-element with a separate power source to keep track of time and date whether or not power is applied to the overall system. In some implementations of hardware clocks, only two digits have been used to report and maintain year data. Unfortunately, a two-digit clock implementation may cause problems as we consider the transition into the next century. If only two digits are returned by the hardware, it may be difficult or impossible for the system element that requested the information to accurately recognize the correct year and century. There are various means to correct for this via firmware or software interpretation, and/or changes to hardware, which this test method does not intend to cover; this test method will attempt to gauge the effectiveness of the remediation, however.

6.6.3 Related elements

Distributed networks

6.6.4 Test cases

6.6.4.1 Hardware system clock transition through crossover.

I Test objective

Verify that the hardware (PC, LAN Server, or Client Workstation) system clock returns the correct date under various power and boot conditions after the Year 2000 rollover.

II Test conditions

#	**Conditions**
1	Enable 8-digit input of date-data. System date-data representation may be displayed in tester's choice, however for consistency sake, ISO 8601 format will be used in this test.

1965 **III Procedure/results**

#	Test procedure	Expected test results
1	Test the system clock automatic update function when the system power is on a. Set the system clock to 1999-12-31, 23:57:00 b. Keep system power on c. Wait 5 minutes. d. Check the date e. If it is set correctly, power off, and re-check the date	Date should be 2000-01-01-for both checks.
2	Test the system clock automatic update function when the power is off Set the system clock to 1999-12-31, 23:57:00 Power the system off Wait 5 minutes. Power the system on Check the date	Date should be 2000-01-01.
3	Test the date/time update after suspension over the crossover of a time-sensitive program that displays the date. Set the system clock to 1999-12-31, 23:57:00 Suspend a date/time-display program without a 'wake-up' timer Wait 5 minutes a. Resume the date/time-display program and check the date	Date should be 2000-01-01.
4	Verify that the system clock, following a powered up crossover, will show the correct date after a warm reboot. Set the system clock to 1999-12-31, 23:57:00 Keep system power on Wait 5 minutes. Check the date Reset (warm reboot) the system and re-check the date	Date should be 2000-01-01 for both checks

1966

6.7 Computer numerical controls

6.7.1 Definition

Computer Numerical Controls (CNC) are system elements that process a program that defines the movements or operation of a machine.

6.7.2 Rationale

Most modern CNCs include a 3.5” floppy drive with associated PC-compatible file system. Fault logging functions which time stamp the occurrence of machine faults are standard features. These time stamps may become corrupted during Year 2000 rollover. Many CNCs are used in an environment where different programs are downloaded for each function performed by the machine. A failure in the file system may cause a file to become inaccessible. In some cases these programs may contain date/time information that the CNC must interpret in order to perform real time functions.

6.7.3 Related elements

Event-triggers, logs-date stamps, file access systems

6.7.4 Test cases

6.7.4.1 Program modification

I Test objective

Verify that modifications to a CNC program are effective after Year 2000 rollover

II Test conditions

#	**Conditions**
1	Prepare a CNC file with a creation/modification date of 1999-12-31
2	Set CNC system date to 2000-01-01

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Open, modify and save the CNC program file.	The CNC program file has a creation/modification date of 2000-01-01
2	Reinitialize the CNC device and check the system date	The CNC system date should remain at 2000-01-01
3	Load and run the modified CNC program.	The CNC program should operate correctly as modified.

6.7.4.2 Logging

I Test objective

Verify that a CNC device can create a log file of events both before and after Year 2000 rollover

II Test conditions

#	Conditions
1	A CNC device with log function.
2	Set system date to 1999-12-31

III Procedure/results

#	Test procedure	Expected test results
1	Enable log function. Generate an event which should cause a log file entry	
2	Wait until after Year 2000 rollover and generate a second event that should cause a log file entry.	
3	Disable the log function and examine the log file created.	The log file should contain entries for two events dated 1999-12-31 and 2000-01-01 in that order.

6.8 Communication protocols

6.8.1 Definition

A set of formal rules describing how to transmit data, especially across a network. Low-level protocols define the electrical and physical standards to be observed, bit and byte ordering, the transmission, error detection and correction of the bit stream. High-level protocols deal with the data formatting, including the syntax of messages, the terminal to computer dialogue, character sets, sequencing of messages etc.

6.8.2 Rationale

Each different communications protocol may have features that depend on date and time. Possible date-dependent areas might include message sequencing over high risk test dates, data routing costs calculations based on date/time and message header date formats. The specific test issues regarding communications protocols must be derived from an analysis of the particular protocol being used. For example, the Simple Network Management Protocol defines two commands (HOLDuntil and MSTAtus) that require date/time date stored using the two-digit year format YYMMDDHHMMSS+GMT. HOLDuntil allows for delayed delivery of a message until after the time specified. Messages with a HOLDuntil after December 31, 1999 may never be delivered, or may be delivered immediately.

6.8.3 Related elements - none

6.8.4 Test cases

6.8.4.1 Time stamps

When email is sent a time stamp is included that may be taken from the system date of the system originating the message. The purpose of this test case is to determine if the time stamp associated with a mail system can be interpreted by the receiving system with the correct century indication. In some cases setting the system date format to six digit may cause the time stamp to be either incomplete or have the digits ‘19’ added to the 2-digit year value.

I Test objective

Verify that correct time stamps will accompany email sent after the Year 2000

rollover. Verify that receiving system correctly recognizes century given two-digit year input.

II Test conditions

#	**Conditions**
1	Email system with at least two registered addresses.
2	Set system time to 2000-01-01
3	Set origination workstation date mode to YY-MM-DD
4	Set receive workstation date mode to YYYY-MM-DD

III Procedure/results

#	**Test procedure**	**Expected test results**
1	On the origination workstation compose and send a message to the receive workstation. Check the time stamp on the message in the sent-mail folder.	The message should be in the sent-mail folder and its date should be 00-01-01.
2	On the receive computer, check the date stamp on the message in the in-box.	The message should be in the incoming-mail folder and its date should be 2000-01-01.
3	Open the message and check the date received.	The date should be 2000-01-01.

6.9 Compilers

There are two distinct aspects of Year 2000 testing with regard to compilers. The first is ensuring that the operation of the compiler itself correctly spans the Year 2000 boundary. The second ensures that the compiler-generated output, i.e., the input source code translated to the target object code, operates correctly across the Year 2000 boundary.

In the first case virtually any source program that can be compiled will serve as a test case. It may be necessary to vary compiler options and/or control statements to demonstrate that date information inserted into compiled programs or displayed on listings or screens is correct. It may be necessary to manipulate the system date for these test cases to be demonstrative.

In the second case special source code must be written to demonstrate that the behavior of the compiler output is correct. It may also be necessary to invoke a variety of pre-processors since those may invoke different library functions for date definition and manipulation. Compilers may have both called library functions for date-related processing and built-in functions where the date-related processing is inserted in-line. All such cases and variations must be examined for correct operation across the Year 2000 boundary.

In both cases the context of the compilation should be considered. There may be date-related considerations which vary depending on whether or not the compiler is part of a development or language environment as opposed to a standalone process. There may also be different considerations if the compiler operates on a system architecture different than the compiled code's target architecture. Database and data communications programs may also require special consideration, in particular if any part of the data or control is distributed where different time zones, date sources or compiler versions may affect the outcome of date-related processing.

6.9.1 Definition

A compiler is a computer program that translates programs expressed in a high-level language into their machine language equivalents.

6.9.2 Rationale

Compilers are an integral part of many software systems. They both translate software program-related dates and process dates to support the functionality of the compiler itself. In the program translation process offsets from a base date and other data-manipulation techniques can be used to aid in conversion. A compiler is similar to any software program that processes a full range of dates with the potential for unexpected failures in date handling.

Compilers may contain date-retrieval facilities that may be invoked by compiled programs. The function of these should be tested. Extra caution may be warranted for compiler pre-processors where the date retrieval mechanism may differ from and override that of the compiler.

6.9.3 Related elements

Date Calculations, parsing

6.9.4 Test cases

6.9.4.1 Compiler program date determination

Compilers generate reports that contain compiler date information. These reports should correctly process and report date information.

I Test objective

Verify that a compiler processes date information and generates date information reports and screen displays regarding its own operation correctly.

II Test conditions

#	**Conditions**
1	A compiler is prepared to accept a high level language file for conversion to machine language. Date-simulation software has been added to the computer system to return compiler calls for system dates beyond the Year 2000.
2	A compiler is prepared to accept a high-level language file for conversion to machine language. The compiler is operating within a computer system whose system clock has been advanced to beyond the Year 2000.

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Record the post-2000 dates that will be returned to the compiler by the date-simulation software	All dates in the compiler reports should be correctly formatted and conform to the post-2000 dates

	and/or system clock. Configure the compiler to output date information as appropriate. Process the file conversion and generate reports to display screen, file, printer and/or other output devices. Inspect the dates in the compiler reports.	returned by the simulation software or future-system environment.

6.9.4.2 Compiler date formatting

The compiler must be able to correctly process all of the date formats that are used by the programs it compiles beyond the beginning of Year 2000 and throughout the compilers valid date interval.

I Test objective

Verify that the compiler is able to read, format, and output date-data in date formats used by the programs it compiles in a Year 2000 environment.

II Test conditions

#	**Conditions**
1	A compiler is prepared to accept a high order language file for conversion to machine language. The high-level language file contains a test program, which reads date-input data, applies specified date formats to the input data, and outputs the date-data. The following are examples of input data and date formats, which may be used by the test program and the output that should result. The form of the input data and the date formats supported may vary among compilers.

Given date	**Date format**	**Date output**
1999-12-31	YYYY/MM/DD	1999/12/31
1999-12-31	YYYY-MM-DD	1999-12-31
1999-12-31	YYYY.MM.DD	1999.12.31
1999-12-31	MM/DD/YYYY	12/31/1999
1999-12-31	MM-DD-YYYY	12-31-1999
1999-12-31	MM.DD.YYYY	12.31.1999
1999-12-31	DDMMYYYY	31121999
1999-12-31	MMM DD,YYYY	DEC 31, 1999
1999-12-31	DD MMM YYYY	31 DEC 1999
1999-12-31	YYYYMMDD	19991231
2000-01-01	YYYYMMDD	20000101
1999-12-31	MMMM DD, YYYY	December 31, 1999
2000-01-01	DDMMMYYYY	01JAN2000
2000-12-31	DDMM	3112
2000-12-31	YYDDD	00366
2000-12-31	MMYY	1200

MMM - Three character abbreviation of the month.
MMMM – Month value displayed as complete word.

III Procedure/results

#	Test procedure	Expected test results
1	Prepare a test program with appropriate date-input data and date formats. Compile the program. Execute the program and inspect the output.	For each input date and date format, the date output should be in the format listed in the Date Output column above.

6.9.4.3 High-risk date translation processing

The compiler should be able to accurately process all high-risk dates that may be included as data within a program.

I Test objective

Verify that the compiler correctly processes all date-related information included in program data.

II Test conditions

#	Conditions
1	A compiler is prepared to accept a high level language file for conversion to machine language. The file to be converted contains the high-risk date-data shown below and a function to add a number of days to each date and print the resulting date. The dates to be in the data include the following: a. 1999-01-01 b. 1999-09-09 c. 1999-12-31 d. 2000-01-01 e. 2000-02-28 f. 2000-02-29 g. 2000-12-31 h. 2001-01-01 i. 2001-02-29 (note: This date should return an error.) j. 2000-10-10

III Procedure/results

#	Test procedure	Expected test results
1	Compile and execute the program. Inspect the output.	Compare the dates that are output with the expected outputs. For example, if 3 days were added to each date, the outputs would be correctly formatted and incremented by 3. If the pre-compiled date was 1999-12-31, the date displayed after the program was compiled and executed would be 2000-01-03.

6.9.4.4 Special-condition date-processing

The compiler should accurately process all special-condition date-data in a program.

I Test objective

Verify that the compiler correctly processes special-condition date-data.

II Test conditions

#	Conditions
1	A compiler is prepared to accept a high-level language file for conversion to machine language. The file to be converted contains a test program which reads pairs of input date-data including special-condition dates, calls a function to calculate the difference between the date pairs and formats, and outputs the pairs of input dates and calculated differences. The following are examples of pairs of input dates involving special-condition dates: First Date Second Date Calculated Difference 1998-12-31 2000-01-02 367 1998-12-31 1999-01-04 4 1999-12-28 2000-01-02 5 1999-12-31 2000-01-10 10 1999-12-31 2000-01-11 11 1999-12-31 2001-01-02 368 2000-02-28 2000-03-02 3 2000-02-29 2000-03-02 2 2000-12-31 2001-01-13 13 2001-01-01 2002-01-01 365 2001-02-29 2001-03-01 (This date should return an error.) 2000-10-09 2000-10-10 1 2000-10-09 2000-10-11 2

III Procedure/results

#	Test procedure	Expected test results
1	Prepare a test program with appropriate date-input data and date formats. Compile and execute the program. Inspect the output.	The input date-data should be output formatted as specified and the calculated differences should be as listed above.

6.10 Event-triggers

6.10.1 Definition

A process which causes the automatic invocation of a procedure at a specified time.

6.10.2 Rationale

Event-triggers generally start the execution of a procedure when the current time is equal to or greater than the scheduled event time. As we approach Year 2000, events will need to be scheduled in 1999 to occur in the Year 2000. If the event-trigger process compares dates with only two digit year information, it might interpret an event flag intended for the Year 2000 as if it were set for 1900. This might cause an error, or the event might occur immediately. Such event flags are often used in systems with built-in maintenance-scheduling elements. A date is stored by these systems at the time that maintenance is performed. This date is compared to the current system date at regular intervals until the next maintenance is due. When the event-trigger goes off, it may only turn on an indicator, but in some systems it might trigger a shut down or intentionally

fuse elements of the system to prevent operation until maintenance is completed. Event-triggers may initiate periodic maintenance of embedded systems.

6.10.3 Related elements

Alarm systems
Project management systems
Invoice aging

6.10.4 Test cases

6.10.4.1 Starting applications

Applications scheduled before the Year 2000 to start after a specified elapsed time should open on schedule even if the event starts after the Year 2000 rollover.

Applications scheduled to start on a specified date after 1999 should open as scheduled.

I Test objective

Verify that an event-trigger process will continue to open applications on a specified schedule beyond the beginning of the Year 2000.

II Test conditions

#	**Conditions**
1	Date/time mode set to YYYY-MM-DDThh:mm:ss with a 24-hour clock.
2	Event-trigger process active.
3	Test application to be triggered has a screen display and requires user to close it upon completion.

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Close test application. Set event-trigger process to open test application every 2 minutes. Wait for test application to open and immediately close it several times to confirm correct operation at current time.	Event-trigger processor should reopen test application every two minutes after it is closed.
2	Set system timer to 1999-12-31T23:57:00. Set event-trigger program to open test application on 2000-01-01 at 00:03:00. Close the test application.	
3	Wait six minutes for test application to open.	Test application should open on 2000-01-01 at approximately 00:03:00.
4	Close test application. Set system timer to 1999-12-31T23:57:00. Set event-trigger program to open test application every 2 minutes.	
5	Wait for test application to open and immediately close it three times.	Test application should open on 1999-12-31 at approximately 23:59:00, and on 2000-01-01 at approximately 00:01:00 and 00:03:00.

6	Repeat steps 2 - 5 putting event-trigger process in background mode after it is set.	Results should be as in steps 2 - 5, unaffected by putting event-trigger in background mode.

6.10.4.2 Sending email

Email written before the Year 2000 rollover but scheduled for transmission afterwards should be sent as scheduled.

Some network systems cache their email and send messages periodically. This should be considered when reviewing expected results.

I Test objective

Ensure that email scheduled after the end of 1999 is sent on schedule.

II Test conditions

#	**Conditions**
1	Operating email system with at least two addresses

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Disconnect transmit system from mail router and attempt to send a message to the receive system.	A message window indicating the message could not be sent should appear.
2	Set system date and time to 1999-12-31T23:55:00. Create a test email message. Choose the Transmit Later Option and set the transmit time to 2000-01-01T00:03:00. Reconnect the system to the mail router. Wait ten minutes. Check for mail at the receive system.	The test message should be in the new message folder.

6.10.4.3 Invoice aging - automatic invoice generation

The life cycle of an invoice is a series of defined periods monitored by the system. Invoices whose life cycle spans Year 2000 rollover will be required to compare dates before and after 1999-12-31. Inaccurate comparisons could lead to invoices that never become payable or are not sent for payment. Other systems may view recent invoices as being 99 years past due.

I Test objective

Verify that invoices created in Dec 1999 will produce billings at 30, 60, 90 and 120 days.

II Test conditions

#	**Conditions**
1	Billing package with at least one current long-term test account that has no outstanding balance.

III Procedure/results

#	Test procedure	Expected test results
1	Set system date to 1999-12-15. Create four invoices (A, B, C, and D) to the test account and four (E, F, G, H) for one-time sales. Print daily outgoing invoices.	All invoices should be printed with a billing date of 1999-12-15.
2	Set system date to 2000-01-14. Print daily outgoing invoices.	All invoices should be printed with a billing date of 2000-01-14 labeled outstanding for 30 days.
3	Set system date to 2000-02-13. Enter “paid in full” for invoices A and E. Print daily outgoing invoices.	Invoices B, C, D, F, G, and H should be printed with a billing date of 2000-02-13 labeled outstanding for 60 days.
4	Set system date to 2000-03-14. Enter “paid in full” for invoices B and F. Print daily outgoing invoices.	Invoices C, D, G, H should be printed with a billing date of 2000-03-14 labeled outstanding for 90 days. Invoices D and H should be printed due upon receipt.
5	Set system date to 2000-04-13. Enter “paid in full” for invoices C and G. Print daily outgoing invoices.	Invoices D and H should be printed with a billing date of 2000-04-13 labeled outstanding for 120 days.

6.10.4.4 Interval timers

Many systems have processes that occur at regular intervals such as checking for new messages every 15 minutes, checking for new data in a shared data source every 5 seconds and polling sensors that tell a machine tool that a new part is available to be processed every few milliseconds. In systems that contain a real time clock (RTC) these and other events may be timed by setting an event-trigger to occur at the current system time plus a specified interval. When the trigger is encountered, a new event-trigger is set and the process is started. When the process completes, the system waits for the next event-trigger and repeats the cycle. During Year 2000 rollover, the computation of the trigger time may produce a non-existent time.

99-12-31T11:59:30 + 5 min. = 100-01-01T00:04:00

In 5 minutes the RTC time of 00-01-01 00:04:00 may not be interpreted as equal to the computed trigger time, so the timer might wait indefinitely.

However, if the system computes a correct date but omits or ignores century digits, then the computed date may be interpreted as before the current time and trigger a new cycle immediately. This might put the timer into a loop condition, in which the timer would restart the process as often as possible until the current time reaches 00-01-01 00:00:00 or the system crashes. For example:

Command	System time	Trigger time
Wait (until System Time ≥ Trigger Time)		99-12-31 23:55:30
Set trigger = System time + 5 min.	99-12-31 23:55:30	00-01-01 00:00:30
Start process (5 sec. Process completion time)	99-12-31 23:55:30	00-01-01 00:00:30

Wait (until System Time ≥ Trigger Time)	99-12-31 23:55:35	00-01-01 00:00:30
Set trigger = System time + 5 min.	99-12-31 23:55:35	00-01-01 00:00:35
Start process (5 sec. Process completion time)	99-12-31 23:55:35	00-01-01 00:00:35
Wait (until System Time ≥ Trigger Time)	99-12-31 23:55:40	00-01-01 00:00:35
Set trigger = System time + 5 min.	99-12-31 23:55:40	00-01-01 00:00:40
Start process (5 sec. Process completion time)	99-12-31 23:55:40	00-01-01 00:00:40
Wait (until System Time ≥ Trigger Time)	99-12-31 23:55:45	00-01-01 00:00:40
The above cycle may repeat until the system time reaches Year 2000 rollover. At that point the 2 digit year values for system time and trigger time become equal and the comparison should function normally.		
Set trigger = System time + 5 min.	00-01-01 00:00:00	00-01-01 00:05:00
Start process (5 sec. Process completion time)	00-01-01 00:00:00	00-01-01 00:05:00
Wait (until System Time ≥ Trigger Time)	00-01-01 00:00:05	00-01-01 00:05:00
Set trigger = System time + 5 min.	00-01-01 00:05:00	00-01-01 00:10:00
Start process (5 sec. Process completion time)	00-01-01 00:05:00	00-01-01 00:10:00

I Test objective

Ensure that an interval timer will maintain its specified cyclic interval when the interval crosses Year 2000 rollover.

II Test conditions

#	**Conditions**
1	Interval timer set to cycle every 5 min.
2	A means of monitoring the timers output
3	System time set to 99-12-31 23:50:00

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Start interval timer at 23:51:00	
2	Observe timer output	Timer should trigger at 99-12-31T23:56:00 and 2000-01-01T00:04:00 (In some implementations the timer may also trigger immediately when it is started)

6.11 Error handling

6.11.1 Definition

The process of detecting and responding to any discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

6.11.2 Rationale

Computer systems routinely process date information. Whether date information is interactively input by the user or whether date-data is supplied via some other source, (e.g., a data file), there often is some means to assess the legitimacy of the date-data. If input date-data is not acceptable to the processing logic, then an error should be reported. As systems or system elements are remediated to accept dates beyond the

Year 2000, the error detection logic and error reporting routines will also need to be remediated. Such remediations should include meaningful error messages.

The use of any dates which can be interpreted as more than one actual date should produce an error or warning. This may provide a notification to the user that it is necessary to correct the stored dates or provide more complete input dates.

6.11.3 Related elements

Archiving/Restoring

6.11.4 Test cases

6.11.4.1 New error messages

In this test case the use of dates outside the test period should produce an error message. Other methods of handling the error may be more suitable to a specific system. For example, the system might mark a date field as unknown or request user intervention to clarify the date. This is acceptable provided that no data corruption occurs. The terminalogy of the test assumes that the term Pivot Year is used in accordance with IEEE Standard 2000.1-1998 sub-clause A.2.4. This limits the range of acceptable dates to 1980-01-01 through 2079-12-31.

I Test objective

Test for appropriate error reporting for input dates that are out of range. Verify that 4-digit years outside the date window for 2-digit database fields will return an error.

II Test conditions

#	**Conditions**
1	An application using a database with at least one date field that has a 2-digit year format and a date window based on 1980 as the pivot year.
2	Set the data entry date format to YYYY-MM-DD

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Create a new record with 1979-12-31 in the date field.	This should cause an error stating that the date 1979-12-31 is out of range.
2	Create a new record with 1980-01-01 in the date field.	The date 1980-01-01 should not cause an error
3	Create a new record with 2079-12-31 in the date field	The date 2079-12-31 should not cause an error
4	Create a new record with 2080-01-01 in the date field	This should cause an error stating that the date 2080-01-01 is out of range.

6.11.4.2 Proper display of dates in error messages

Error messages that contain dates must display them in a format that reliably differentiates the century digits.

I Test objective

Verify that the error message produced when attempting to delete an archive before its expiration date displays the date in an unambiguous manner.

II Test conditions

#	Conditions
1	A system with an archive function that allows a user to set the archive retention period.

III Procedure/results

#	Test procedure	Expected test results
1	Set System date to 1998-06-06	
2	Create archive A with 1-year data retention	
3	Create Archive B with 2-year data retention	
4	Set system date to 1998-12-01	
5	Attempt to delete archive A	Should get an error message stating that archive should be retained until 1999-06-06
6	Attempt to delete archive B	Should get an error message stating that archive should be retained until 2000-06-06

6.12 File access system

6.12.1 Definition

The hardware, software and firmware responsible for the long term storage and retrieval of data and program information.

6.12.2 Rationale

Many systems associate a date with each file. Applications may access these dates to select or compare files. System logic may provide warnings when an attempt is made to replace a file with one that has an older creation date. Other systems keep a specified number of older versions of files as backups. If the process that triggers these features does not properly disambiguate the date the system may fail.

6.12.3 Related elements

Archiving/restoring, Backup and restore

6.12.4 Test cases

6.12.4.1 Application installation

When reinstalling an application, date comparisons may be made to insure that configuration files that have been changed by the user are not replaced. If the file system provides incomplete date information, files modified after the transition from 1999 to 2000 may be returned to an initial configuration.

I Test objective

Verify that reinstallation of software after the transition from 1999 to 2000 will not overwrite configuration information in files dated before the transition from 1999 to 2000.

II Test conditions

#	Conditions
1	System date set prior to Year 2000

III Procedure/results

#	Test procedure	Expected test results
1	Perform normal initial installation of application under test. Note the modification dates of configuration files.	Modification dates should be the current system date or earlier dates.
2	Change the system date to a date after the transition from 1999 to 2000.	
3	Make any modifications necessary to create a change in each configuration file. Note the new configuration file Modification dates	Modification dates should be the new system date after change from step 2.
4	Reinstall the application. Choosing any options offered to retain existing configuration.	Configuration file modification dates should be the same as in step 3.

6.12.4.2 Source code or document control

As a version-control system moves from 1999 into 2000, files created in 1999 or earlier may need to be replaced with files having creation dates in 2000. If this comparison is made using a 2-digit year, the version control system may disallow or inhibit the storage of new files. Files with modification dates prior to Year 2000 might replace newer files without warning.

I Test objective

Verify that retrieval of a pre-2000 file into a directory containing an earlier version of the same file produces a warning.

II Test conditions

#	Conditions
1	Set system date on client and server to 1999-12-01. Create a file on client.
2	Set system date to 2000-01-01
3	Move file to a centralized server whose date is 2000-01-01

III Procedure/results

#	Test procedure	Expected test results
1	Create and save file on client.	
2	Upload file to server.	
3	Change client and server system dates to 2000-01-01	
4	Modify and save file on client (using same file name).	
5	Upload modified file to server to same location as original (unmodified) file.	
6	Delete modified file from client.	

7	Download modified file from Server back to client and open	File on client reflects modified data and also new date.

6.12.4.3 Data upload

When a system or system element sends data to another system or system element, the creation dates of one or more files may need to be compared in order to insure that more recent data will not be overwritten

I Test objective

Verify that a warning message will be returned if an attempt is made to upload a file created before Year 2000 and replace a file dated after Year 2000 rollover.

II Test conditions

#	**Conditions**
1	Set system date on client and server to 1999-12-01.

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Create and save file on client.	
2	Upload file to server.	
3	Change client and server system dates to 2000-01-01	
4	Modify and save file on client (using same file name).	
5	Upload modified file to server to same location as original (unmodified) file.	
6	Delete modified file from client.	
7	Download the file from the server back to client and open it.	File on client reflects modified data and also new date.

6.12.4.4 File overwrite

Many systems enhance data safety by comparing file dates and providing warnings if a system element attempts to overwrite an existing file with an earlier version of the same file. These warnings may occur when not needed if the existing file was created before and the replacement file is created after Year 2000 rollover.

I Test objective

Verify that a file created after Year 2000 will replace a file dated before Year 2000 rollover.

II Test conditions

#	**Conditions**
1	Existing files dated before Year 2000 rollover
2	System date set after Year 2000 rollover.

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Open the existing file and make some changes to it.	
2	Save the file to a different location in the file system.	File is saved without error

3	Restart the system that uses the file.	
4	Open the file saved in step 2.	
5	Save this file to the location of the original test file using the same name.	Either no warning should occur or the file being saved should be listed as the newer file.

Globalization/internationalization

6.13.1 Definition

The process of modifying or developing a system that has an interface that is not limited to the date/time information formats used by any specific country or social group.

6.13.2 Rationale

An application may be effectively remediated for one date format and yet fail when another regional date format is required. The lack of a globally implemented date standard complicates the issue of Year 2000 conversions. The order of date and time information, separators used and different calendars all require conversions and special logic. Any of these functions may distort or render the century digits ambiguous. ISO 8601 presents an international standard format for date and time information. Adherence to this standard should reduce confusion and allow greater portability of the system interface. However, for the great number of systems that were constrained by the requirements of interfacing to the installed systems that pre-date ISO 8601 and therefore do not use that standard, efforts must be made to interface reliably to them as they are. Also, remediated systems may not adhere to ISO 8601 due to the impracticality of year expansion and may present nonstandard formats to interfacing systems. Operating systems generally provide the basic interface functions to accept and display data. Applications must in turn provide complete, properly formatted information to the operating system. The overall system must in turn be able to exchange data in agreed upon formats acceptable to systems with similar setting as well as systems set for completely different regions in different languages. Even systems that do not span cultural boundaries can have similar interfacing problems. For instance, a remediated system with a 2-digit format with one pivot year may have to interface with another remediated system with a 2-digit format that uses a different pivot year. Embedded systems generally must confront the same issues when interfacing with display modules and with other systems.

6.13.3 Related elements - none

6.13.4 Test cases

No specific test cases will be referenced here in their entirety, rather, test cases will be referenced and suggestions made as to how they may be adapted to suit globalization and internationalization test objectives. Any other date specific globalization/localization issue may need to be tested.

Sub-clause	Content	Modification to test conditions or test procedure
6.26	Sort	Sort-control information should match the date format to achieve the expected result.
6.23	Query	Query-control information should match the date format to achieve the expected result.
6.28	Date format	Adapt the test conditions, procedures, and expected results date format to coincide with each localized environment. For example, U.S. may use MM-DD-YY and a European country may use DD-MM-YY.

6.4.4.2 6.4.3	Holidays	Holidays will vary with the specific region or country of origin. For example, July 4 may be a recognized U.S. holiday, but it is not similarly celebrated in other nations.
6.20	Parsing	Parsing-control information must match the date format to achieve the expected result.
6.11	Error Trapping	Error trapping must match the date format to achieve the expected result. For example an error trap may trap the year in two digits and the remediation changes the date format to 4-digit format; the trap should also be adjusted to match the 4-digit year. Or, an error may trap the year in two digits and remediation keeps the year format in two digits but the system now has an implicit pivot year.

6.14 Synchronization of distributed networks

6.14.1 Definition

Time and date synchronization is the state at which time information from separate nodes in a distributed network is equalized within some defined tolerance.

6.14.2 Rationale

Time and date synchronization over a distributed network may be a part of normal operation. Distributed networks transfer time and date-data in different manners. One is to periodically update other elements within the distributed network. An example of this is a primary or reference timeserver updating secondary servers across the multi-server network.

Another manner in which date-data is distributed or updated is when network elements, such as workstations, set the time and date upon specific tasks. One example of such tasks would be the login process; if appropriately configured, a server can then set a workstation's date and time. Needless to say, the workstation must be capable of accommodating date-data beyond 11:59PM on 1999-12-31.

Tests described in this section attempt to validate network element's capabilities of setting and accommodating date-data beyond the Year 2000.

6.14.3 Related elements :

Tests for stand-alone hardware

6.14.4 Test cases

6.14.4.1 Primary and/or reference time servers and secondary servers.

I Test objective

Verify changing system time on primary or reference timeserver creates a corresponding change in connected secondary servers

II Test conditions

#	Conditions
1	Network with one primary or reference server connected to multiple secondary servers
2	Set date mode to an 8-digit data.
3	Servers are in same time zone

2393
2394 **III Procedure/results**

#	Test procedure	Expected test results
1	Set the primary or reference server to 1999-12-31, 23:45:00 and check the time on all secondary or replicated servers.	All clocks should match the primary or reference server's.
2	Wait for the primary or reference server date to reach 2000-01-01. Check the time on all secondary systems in the cell. The time interval required before checking all secondary servers depends on many factors such as (but not limited to) the processor load and performance of the primary or reference server, the quantity and types of any remote sites, and the number of client workstations active on the network.	All clocks should match the primary or reference server's.
3	Reboot the server and check its system time	System time should be approximately the same before and after the reboot.
4	Check each secondary server's clock. Reset the secondary servers and check system time.	All clocks should match the primary or reference server's, before and after the reset.

2395

2396 **6.14.4.2 Servers and workstations**

2397 **I Test objective**

Verify changing system time on a server will create a corresponding change to client workstations during the next user logon.

2398
2399 **II Test conditions**

#	Conditions
1	LAN with one server connected to multiple client workstations.
2	Set date mode to a format that displays a 4-digit year.
3	Server is configured to update workstation time and date information.
4	Server and workstations are in same time zone.

2400

III Procedure/results

#	Test procedure	Expected test results
1	Set the server to 1999-12-31T23:45:00.	
2	Set the client workstations clocks and calendars to 1999-06-01T12:00:00.	
3	Logoff if necessary and power off the client workstations.	
4	Watch the server transition to 2000-01-01.	
5	Power up the client workstations and login to server.	
6	Check the client workstations time and date.	Upon completion of step 6 the clocks of all client workstation's should match that of the server

6.14.4.3 System synchronization over multiple time zones.

This test case checks a variation of a problem that occurs daily on systems that reside in different time zones. Normally date transition would be separated by one or more hours for such systems. In the 1999/2000 transition case, date transition may cause differences of 100 years for a period of one or more hours. This may invalidate controls on checking for acceptable data in transfers and may cause transactions to be rejected erroneously.

Remember that not all time zones are offset in whole hour increments. If the system may be required to transmit data to and from time zones that have an offset expressed with a fraction of an hour, the test case may need to be repeated for situations where one or both of the systems are set to a fractional time zone.

It is also important to remember that not all instances of Daylight Savings Time (DST) or "Summer Time" are increments of whole-hours. For global businesses having processing nodes in the Southern Hemisphere, the 1999/2000 transition will happen during this "Summer Time".

For these tests it is important to recognize that time zone information may be stored independently of the time setting in the system. In fact, this is normally the case. This test case will be ineffective if the time zones are not set properly. This is particularly important for the case of non-integral hour time zones where the time zone difference will be some number of hours plus a fraction.

I Test objective

Verify that the date difference caused by the Year 2000 transition has no effect greater than the one-day date difference caused by the normal daily date transition between systems in different time zones.

II Test conditions

#	**Conditions**
1	System A with a system date and time set to 1999-12-31T23:50:00 in any arbitrary time zone.
2	System B with a system date and time of 1999-12-31T22:50:00.
3	System A and B are capable of communicating across a network.
4	System or application software that is sensitive to date/time differences greater than one day.

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Create test data on system A	
2	Transmit test data to system B	Transmission should occur normally and data should be correct
3	Allow system A to roll over into Year 2000	
4	Repeat steps 1 and 2	Transmission should occur normally and data should be correct
5	Create test data on system B	
6	Transmit test data from step 5 to system A	Transmission should occur normally and data should be correct

6.14.4.4 System synchronization over multiple time zones (continued)

This test case is designed to verify the ability of a system to cope with the Year 2000 rollover across time zones that are separated by a non-integral number of hours. This can occur because the normal time zone offset is a non-integral number of hours from UTC or because the Daylight Savings Time or "Summer Time" adjustment is not a whole hour. Both cases occur across the globe and should be of special concern to multinational enterprises.

I Test objective

Verify that the date difference caused by the Year 2000 rollover has no effect greater than the one day date difference caused by the normal daily date transition between systems in different time zones where those time zones are offset by a non-integral number of hours. The example given is artificial, but a real-life example can be constructed using time zones in the United States for one system and time zones in parts of northern or southern Australia for the other. Refer to a table of international time zones for specifics if needed.

II Test conditions

#	Conditions
1	System A with a system date and time set to 1999-12-31T23:50 in any arbitrary time zone with an integral number of hours offset from UTC, e.g. UTC + 8.0
2	System B with a system date and time of 1999-12-31T22:50 in a time zone with a non-integral numbers of hours offset from UTC, e.g. UTC + 7.5

3	System A and B are capable of communicating across a network.
4	System or application software that is sensitive to date/time differences greater than one day.

III Procedure/results

#	Test procedure	Expected test results
1	Create test data on system A	
2	Transmit test data to system B	Transmission should occur normally and data should be correct
3	Allow system A to roll over into Year 2000	
4	Repeat steps 1 and 2	Transmission should occur normally and data should be correct
5	Create test data on system B	
6	Transmit test data to system A	Transmission should occur normally and data should be correct
7	Allow system B to roll over into Year 2000	
8	Repeat steps 1, 2, 5 and 6	Transmission should occur normally and data should be correct.

6.15 Import/export

6.15.1 Definition

The processes of converting one file format to another in order to use data in more than one system.

6.15.2 Rationale

After remediation, and depending on the remediation used, two systems may be unable to exchange date data, or date data may be exchanged but the meaning of the dates may be different in each system. The century interpretation logic may not be accessible to the receiving application.

6.15.3 Related elements

Multi-System Windowing

6.15.4 Test cases

6.15.4.1 Database conversion exchange

Date values of original files will retain and express the correct century-digit information after conversion to a new data format.

I Test objective

Ensure that importing a database from an application that uses a sliding date window to one that uses an eight-character date will produce equivalent data in both applications.

II Test conditions

#	**Conditions**
1	Set system time to 1999-01-01
2	Application A that requires an 8-digit date field.
3	Application B that uses a database with a 6-digit date field and a sliding date window from 1930 to 2029 set to increment the pivot year at the end of each year.
4	A database for application B with records having the following date fields. 1930-01-01 1999-12-31 2000-01-01 2029-12-31

III Procedure/results

#	**Test procedure**	**Expected test results**
1	In application A import the data base from application B. Open the database and check each record date field.	The following dates should be present. 1930-01-01 1999-12-31 2000-01-01 2029-12-31
2	Set the system time forward 1 year. Wait for database in application B to update. Check each record date field.	The following dates should be present. 1999-12-31 2000-01-01 2029-12-31
3	In application B add records with the dates 1931-01-01 and 2030-12-31. Then sort by date, and check the results.	The following dates should be present. 1931-01-01 1999-12-31 2000-01-01 2029-12-31 2030-12-31
4	In application A import the database from application B. Open the database, and check each record date field.	The following dates should be present. 1931-01-01 1999-12-31 2000-01-01 2029-12-31 2030-12-31

6.15.4.2 Maintaining database integrity

If data is transferred from a source that has a wider valid date range than can be represented in the format of the destination database it may be impossible to transfer some date-data to the destination database. In the case below 8-character dates must be transferred into 6-character fields. Using the standard 6-character data format limits the destination database to storing dates within a 100-year window. If dates outside this window are not detected and processed separately, the system is likely to corrupt the database by erroneously changing these out of range dates to dates that fall within the window. Depending on the design of the database, the system may eliminate records containing out of range dates, produce an error and not allow the database to be transferred, request that the user enter a value within the range, etc. The system specification and other documentation should define how dates outside the current window will be represented in the database. In this test case it is assumed that out of range values are automatically replaced with an indicator value that the system displays as “Out of Range”.

I Test objective

Ensure that importing a database from an application that uses a 2-character year to one that uses a sliding date window will produce equivalent data in both applications

II Test conditions

#	**Conditions**
1	Set system date to 1999-01-01
2	Application A that requires an 8-digit date field.
3	Application B that uses a database with a 6-digit date field and a sliding date window from 1930 to 2029 set to increment the pivot year at the end of each year.
4	A database for application A with records having the following ID and date fields. ID Date 1 1929-12-31 2 1930-01-01 3 1999-12-31 4 2000-01-01 5 2029-12-31 6 2030-01-01

III Procedure/results

#	**Test procedure**	**Expected test results**
1	In application A export the data-base to application B. Open the database in application B and check each records date field.	The following records should be present. ID Date 1 Out of Range (see note) 2 1930-01-01 3 1999-12-31 4 2000-01-01 5 2029-12-31 6 Out of range (see note) Note 1: These records contain dates that fall outside application B’s valid date range. They may be modified in other ways than shown as described in the introduction of this test case.

2	Set the system time forward 1 year. Wait for application B's database to update. Check each record's date field.	The following records should be present. ID Date 1 Out of Range (see note 1) 2 Out of Range (see note 2) 3 1999-12-31 4 2000-01-01 5 2029-12-31 6 Out of Range (see note 3) Note 2: This record moved outside the sliding window as the pivot year changed Note 3: Even though the date in record 6 on application A is now within application B's valid date range. It may still be represented as Out of Range until it is updated from database A.
3	In application A add records with the dates 1930-12-31, 1931-01-01, 2030-12-31 and 2031-01-01. Then sort by date and check the results.	ID Date 1 1929-12-31 2 1930-01-01 3 1930-12-31 4 1931-01-01 5 1999-12-31 6 2000-01-01 7 2029-12-31 8 2030-01-01 9 2030-12-31 10 2031-01-01
4	In application A export the data base to application B. Open the database in application B and check each records date field.	The following records should be present. ID Date 1 Out of Range (see note 1) 2 Out of Range (see note 1) 3 Out of Range (see note 1) 4 1931-01-01 5 1999-12-31 6 2000-01-01 7 2029-12-31 8 2030-01-01 9 2030-12-31 10 Out of Range (see note 1)

6.15.4.3 File import/export

During the remediation process it may be necessary to change the date storage format to allow for the representation of post-Year 2000 dates within a system or system element. If a file is exported from this remediated system the file's date format may need to be extended to allow these dates to be interpreted. It is also possible that the system that will process the file may misinterpret dates due to the change in format or date range.

I Test objective

Ensure that exported formatted date data date-data from a remediated system will be properly imported into a secondary system

II Test conditions

#	Conditions
1	Two systems (or system elements) with date set beyond 1999-12-31

III Procedure/results

#	Test procedure	Expected test results
1	Create a file to be exported which contains various date-data. The date-data should be pre- and post Year 2000 rollover.	
2	Export file.	
3	Import file into secondary system.	Pre- and post Year 2000 rollover data is accurately represented.
4	Export file from secondary system and import into first.	Pre- and Post Year 2000 rollover data is accurately represented.

6.16 Multi-system windowing

6.16.1 Definition

Multi-system windowing pertains to the interoperability between two systems or system elements using windowing techniques especially with respect to the transference of date-data between them.

6.16.2 Rationale

Although two remediated systems may separately operate normally, there is potential for error during data transfer between them. If one of the systems is using a date windowed technique to determine century information, then the two systems may need to agree on what the pivot year will be. If they agree, the transmission of dates between the systems must then be limited to the 100 year span determined by the pivot year. If both systems have windows with different pivot years, the date-data must be within the intersection of those windows, unless the window information is also transmitted as part of the interface. For example, a motor vehicle registration office might not be concerned with traffic records over 7 years old so a pivot year of 1990 would give them until 2089 to change to four digit year codes. The system that keeps track of drivers license renewals would have to work with birth dates back to the early 1900s so 1905 to 2005 might be a good date window. In this case, the exchange of date information would be limited to 1990 to 2005. Outside this range, dates would be misinterpreted by 100 years.

6.16.3 Related elements

6.16.4 Test cases

6.16.4.1 Application to application conflict of valid windows.

I Test objective

Verify handling of date-data between two systems is not erroneous due to conflict of valid windows with different pivot years.

II Test conditions

#	**Conditions**
1	Two systems capable of accommodating the Year 2000 rollover.
2	Each system or system element must be capable of unidirectional or bi-directional communication.
3	Each system must have differing pivot years where window information is not transmitted as part of the interface.

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Determine pivot year for each application. If not found in documentation, The pivot year can be found by generating various test cases (using 2-digit years) and determining at which point the century information changes from 19XX to 20XX.	
2	Find Date Ranges: Compare the pivot years for each system. Determine which dates are commonly agreed upon between the systems and which are not. i.e., when comparing the date ranges of each system: a) Common Windows: The set of all 6-digit dates for which both windows will interpret the date to an equivalent 8-digit date b) Conflicting Windows: The set of all 6-digit dates for which the two windows will interpret the date as an 8-digit date on different sides of the 1999/2000 boundary.	
3	Generate a test file that contains date-data commonly on both sides of the 1999/2000 boundary as well as data that is in the conflicting range.	
4	Transfer the test file from one system to another and inspect the data.	

5	Repeat step 4 but transfer data in the opposite direction if bi-directional communications transfers are supported.	The date interpretation of the first system should match that of the second.. Year data that is in the conflicting window should either show an error message or should show a date in a predicted manner (i.e., as predicted to be in either 19XX or 20XX).

6.17 Licensing

6.17.1 Definition

Licensing procedures automatically enable and disable a system, in accordance with the terms of a legal agreement between the developer and the customer. Not all licenses are permanent. In some cases the system may restrict customer access to all or part of the system after a specified date, or after a period of time from installation has elapsed. These systems often encode dates into license keys that can be used to set license expirations .

6.17.2 Rationale

If the term of the license extends across the Year 2000 rollover then subtracting the dates will return a value that can never reach the desired termination. Consider a 5-year license starting in 1997. If the range of years for the system is 00 to 99 there is no number in the range that 97 can be subtracted from to yield 5, so the trigger can not occur. On the other hand, adding 5 to the current date's 2-digit year may yield a value of 102, which is also out of range.

6.17.3 Related elements

Event-triggers

6.17.4 Test cases

6.17.4.1 Termination of application license

In some instances expiration of a license occurs after the correct actual lapsed time from the installation. This lapsed time should be consistent regardless of 1999/2000 transition.

I Test objective

Verify that license expiration will occur on schedule after the Year 2000 rollover.

II Test conditions

#	**Conditions**
1	An application that will return an error when opened if the license is expired.
2	License keys for 30 days, 365 days, permanent and terminating on the following dates. 1999-09-09 1999-12-31 2000-01-01 2000-02-29 2001-02-28
3	System clock set to 1999-02-15.

2559
2560 **III Procedure/results**

#	Test procedure	Expected test results
1	Install a copy of the application with a permanent key, open and close it.	The application should open normally
2	Change the system time to 1999-09-08T23:59:00. Wait for 2 minutes, then open and close the application.	The application should open normally
3	Change the system time to 1999-12-31T23:59:00. Wait for 2 minutes, then open and close the application.	The application should open normally
4	Change the system time to the last day in the test period 23:55:00. Open and close the application.	The application should open normally
5	Uninstall the application. Check for its associated directories and known modifications to system files. Attempt to run the application.	Uninstall should leave no trace of the application. It should not open or return any messages that are not operating-system generated.
6	Set system time to 1999-06-06. Install a copy of the application with a key that expires on 1999-09-09, open and close it.	The application should open normally
7	Change the system time to 1999-09-08T23:59:00. Wait for 2 minutes, then open and close the application.	The application should open normally
8	Change the system time to 1999-09-09T23:59:00. Wait for 2 minutes, then open and close the application.	The application should display a license expiration message and not open.
9	Change the system time to 1999-12-31T23:59:00. Wait for 2 minutes, then open and close the application.	The application should display a license expiration message and not open.
10	Uninstall the application. Check for its associated directories and Known modifications to system files. Attempt to run the application.	Uninstall should leave no trace of the application. It should not open or return any messages that are not operating-system generated.
11	Set system time to 1999-12-15. Install a copy of the application with a 30-day key, open and close it.	The application should open normally
12	Change the system time to 1999-12-31T23:59:00. Wait for 2 minutes, then open and close the application.	The application should open normally

13	Change the system time to the last day the license should be good for 23:55:00. Open and close the application.	The application should open normally
14	Wait for 6 minutes, then open and close the application.	The application should display a license expiration message and not open.
15	Repeat steps 10 - 14 for all remaining keys.	Same results as 10 - 14 for all remaining keys.

6.18 Logs/date stamps

6.18.1 Definition

The creation of a file containing a record of transaction information over a time period.

6.18.2 Rationale

The purpose of logs and date stamps is to give us a way to track events where order of occurrence is important. They are often used to locate errors and to recover to a specific time. If the date and time information contained in logs becomes inaccurate as the system or system element transitions into the Year 2000, then the basic purpose is lost. Examples:

Operating systems
Network software
Email server software
Database Systems

6.18.3 Related elements

Synchronization
Communications
Database recovery

6.18.4 Test cases

6.18.4.1 Server log file

I Test objective

Ensure that server log file dates will be correct after Year 2000 rollover.

II Test conditions

#	**Conditions**
1	Server with event logging enabled

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Set the system date and time to 1999-12-31T23:55:00. Reboot the server. Open the events log and check the entry indicating the logging process was started.	The time date stamp should be 1999-12-31 between 23:55:00 and 23:59:59.
2	Wait for the system date to change to 2000-01-01. Reboot the server. Open the events log	The date stamp should be 2000-01-01.

	and check the entry indicating the logging process was started.	

6.18.4.2 End user log file

I Test objective

Ensure that workstation log file dates will be correct after the Year 2000 rollover.

II Test conditions

#	**Conditions**
1	Workstation with event logging enabled

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Set the system date and time to 1999-12-31 23:55:00. Create a situation that results in an entry in the log being tested. Open the events log and check the entry for the latest entry	The time date stamp should be 1999-12-31 between 23:55:00 and 23:59:59.
2	Wait for the system date to change to 2000-01-01. Reboot the workstation. Open the events log and check the entry for the latest entry	The date stamp should be 2000-01-01.

6.19 Merge

6.19.1 Definition

To combine two or more data sources into a data file with a specified organizational structure.

6.19.2 Rationale

Data stored in multiple files may contain dates both before and after the turn of the century and dates stored in different formats. Merge routines must be able to interpret and distinguish these dates in order to merge records correctly. A database may store dates with a 4-character year, but the merge procedure of a system may still not properly interpret them. Some systems may use a single merge procedure for both 6 character and 8 character date keys by ignoring the top two digits of the 8-character date's year, effectively creating a 100-year window. When the date keys span Year 2000 rollover, the dates 2000-01-01 and after may be interpreted as earlier than dates before rollover.

6.19.3 Related elements

Archive restoration
Databases
Multi-System Windowing

6.19.4 Test cases

6.19.4.1 Non-conflicting date formats

I Test objective

This test determines the ability to merge two different data sources with the same data formats across the Year 2000 boundary.

II Test conditions

#	Conditions
1	Two data files using the same date format
2	One data file contains date-data that spans across the Year 2000 boundary.
3	The second data file contains date-data which doesn't span the Year 2000 boundary
4	Data files must be sorted with key data, which are the dates in question. These may include boundaries of the valid date intervals, dates close to Year 2000 rollover, and other high risk dates associated with the system.

III Procedure/results

#	Test procedure	Expected test results
1	Merge the two data files into a third data file	Resultant data file is in appropriate date order.

6.19.4.2 Conflicting date formats

When merging two databases that have the same fields, instances may occur where older date-data may be in 6-character format and the newer data is in 8-character format.

I Test objective

This case verifies that an ascending order merge sort results in a correctly dated and ordered output list in 8 character format when data includes both 6 and 8 character dates and spans two centuries.

II Test conditions

#	Conditions
1	Database A using an 8-character date format with the following entries in the date fields. YYYY-MM-DD 1900-01-01 1997-07-07 1999-12-31 2044-04-04 2100-12-31
2	Database B using a 6-character date with a pivot year 1920 and the following entries in the date fields. YY-MM-DD 20-01-01 77-07-06 99-12-31 00-01-01 19-12-31

III Procedure/results

#	Test procedure	Expected test results
1	Merge database A with database B.	
2	Sort the merged data in ascending date order.	
3	Store the result in a new file C in 8-character date format.	File C should be created.
4	Open file C and view its content.	Records should be in the following order and dates should be in a format that shows the century. (Note that the ordering of 6 character and 8 character formats equivalent to the same date is immaterial for these purposes.) 1900-01-01 1920-01-01 1977-07-06 1997-07-07 1999-12-31 1999-12-31 2000-01-01 2019-12-31 2044-04-04 2100-12-31
5	Repeat steps 1 to 3, sorting the merged data in descending date order in step 2.	Records should be in the same format but in reverse order from the result for step 4 above.

6.19.4.3 Merge-link

The merge-link procedure used in the test case creates one database from two source databases linked by a shared field. The resulting database contains fields from both original databases. The shared date field has one entry for each discrete value from either source database. Records for dates that appear in both source databases should have values for all fields that had values in either source. Dates that appear in only one source database will have null values for the fields that exist only in the other database. This is not the only possible implementation for a merge-link. The results of the test may vary depending on the specific implementation. The results for a specific system should be repeatable, useful within the context of the system's purpose and consistent with results for a similar test where all dates are prior to Year 2000.

I Test objective

When merging two databases that are linked by dates, and the dates in one database are in 6-character format and those in the other are in 8-character format, verify that the results of a merge correctly match data fields for the same date.

II Test conditions

#	Conditions
1	Database A using an 8-character date format with the following entries in its data and date fields. FIELD1 YYYY-MM-DD 1 1899-12-31 2 1900-01-01 3 1919-12-31 4 1920-01-01 5 1977-07-07 6 1999-12-31 7 2000-01-01 8 2019-12-31 9 2020-01-01 10 2099-12-31
2	Database B using a 6-character date with a pivot year 1920 and the following entries in its data and date fields. FIELD2 MM-DD-YY A 01-01-20 B 07-06-77 C 12-31-99 D 01-01-00 E 12-31-19

III Procedure/results

#	Test Procedure	Expected test results
1	Merge-link by date-database A with database B and store the result in a new database C.	Database C should be created.
2	Open database C and view its content.	Database C contains the following records. Field1 Date Field2 1 1899-12-31 2 1900-01-01 3 1919-12-31 4 1920-01-01 A 1977-07-06 B 5 1977-07-07 6 1999-12-31 C 7 2000-01-01 D 8 2019-12-31 E 9 2020-01-01 10 2099-12-31

6.20 Parsing/validation

6.20.1 Definition

Dividing a stream of data into individual objects that can be easily interpreted by the system.

6.20.2 Rationale

In the process of converting input into arguments to be passed to procedures, the century data may be discarded by parsing functions. Testing of parsing functions

ensures that all input information is available to computational and storage functions. Some parsing functions may convert input data containing only two digits of year information by supplying assumed century data, perhaps as part of a windowing procedure. Some parsing functions may also validate input data, e.g., by checking for correct recognition of leap year days (2000-02-29) and rejecting invalid or ambiguous dates (2000-02-30, 2000-13-32, 03-02-04). Validation is sometimes performed as a separate procedure following initial parsing of input, and the test cases below may be adapted for use in those circumstances.

6.20.3 Related elements

Windowing, Compilers

6.20.4 Test cases

This test case assumes that the Parsing/Validation system element may be unit tested through a debugger or test scaffolding that allows direct manipulation of inputs and interpretation of its outputs. In this particular design outputs retain the values of the last valid input. Other designs might have different post conditions; i.e. setting the output variables to a null value.

6.20.4.1 Screen input

I Test objective

Verify that date-input function correctly parses information the user enters on screen into individual date variables.

II Test conditions

#	**Conditions**
1	A program for date-input function for on screen entry.
2	Date format set to YYYY-MM-DD
3	Screen input is parsed and stored as 4 character year, 2 character month, and 2 character day.

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Enter 1999-12-31 as date	Variable values should be YYYY=1999, MM=12, andDD=31
2	Enter 2000-01-01 as date	Variable values should be YYYY=2000, MM=01, and DD=01

6.20.4.2 Date retrieval

I Test objective

Verify that date retrieval function correctly retrieves 8-character date information from database records.

II Test conditions

#	**Conditions**
1	Database program setup for retrieving records with date key information.
2	Date format set to YYYY-MM-DD
3	Test database for each supported date format with date field records with the following values: 1999-21-31 2000-01-01
3	Date-data is parsed and stored as 4 character year, 2 character month, and 2 character day.

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Retrieve date from first record of the database.	Variable values should be YYYY=1999, MM=12, DD=31
2	Retrieve date from second record of the database	Variable values should be YYYY=2000, MM=01, DD=01

6.20.4.3 Screen input validation

I Test objective

Verify the date-input function correctly validates date information entered by the user on the screen

II Test conditions

#	**Conditions**
1	Program for date-input using on screen date entry, which validates that input consists of only valid dates.
2	Date format set to YYYY-MM-DD

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Enter 1999-12-31 as a date	No error indicated
2	Enter 199P-12-31 as a date	Meaningful Error message should be given
3	Enter 1999-1P-31 as a date	Meaningful Error message should be given
4	Enter 1999-12-PP as a date	Meaningful Error message should be given.
5	Enter 2000-01-01	No error indicated
6	Enter 2000-02-29	No error indicated
7	Enter 2001-02-29	Meaningful error message should be given
8	Enter the date 2000-00-10	Meaningful error message should be given
9	Enter the date 2000-13-10	Meaningful error message should be given
10	Enter the date 2000-09-31	Meaningful error message should be given
11	Enter the date 2000-09-00	Meaningful error message should be given

12	Enter the date 99-12-31	Meaningful error message should be given
13	Enter the date 00-01-01	Meaningful error message should be given

6.21 Performance

6.21.1 Definition

Operation's effectiveness as measured by a comparison of task completion times or system resources necessary for task completion.

6.21.2 Rationale

The speed of queries, sorts, and computations may all be adversely affected by changes necessary to handle dates beyond Year 2000 rollover. Changing data formats may increase drive space necessary to store date-data. If documentation of performance specifications exists, the tests used to verify performance claims should be repeated to verify the accuracy of these figures. For better accuracy tests may be changed and tests conducted using current dates, rollover transition dates, and dates after Year 2000 rollover. Test data should be aged to make it consistent with expected data for the time frame being tested.

6.21.3 Related elements - none

6.21.4 Test cases

6.21.4.1 Performance regression

Performance claims can be maintained across the Year 2000 rollover. Data that spans two centuries will not cause unwarranted degradation of performance.

I Test objective

Verify that performance will stay within published specifications.

II Test conditions

#	**Conditions**
1	Standard test environment with a standard set of test data that includes date information.
2	There should be two additional sets of test data. One that has had the date information modified to include a similar number of dates falling on either side of the Year 2000 rollover. The second should hold dates only after rollover. Adding a number of weeks to the original test data can, in some instances, create these data sets.

III Procedure/results

#	**Test Procedure**	**Expected test results**
1	Perform a standard performance test as outlined in current testing procedures. Check results	Results should match published performance claims.
2	Change the resource to use the second set of test data. Modify the system time by advancing it the same number of weeks the test data was advanced. Perform performance test as outlined in	Results should match published performance claims.

	current testing procedures. Check results	
3	Change the resource to use the third set of test data. Set the system time to the original test system time plus the number of weeks the test data was advanced. Perform performance test as outlined in current testing procedures. Check results	Results should match published performance claims.

6.22 Operational time periods

6.22.1 Definition

The interval of time required for a cyclic phenomenon to complete a cycle and begin to repeat itself.

6.22.2 Rationale

Accounting and scheduling applications often declare blocks of time known as periods to group sets of data. If these sets can no longer be created or the information they contain can no longer be accurately manipulated the application will fail. Tests that insure a period has a valid start date (One that was prior to its end date.) may fail when the period spans the Year 2000 Rollover. There may also be problems testing what information belongs in a specific period.

6.22.3 Related elements

Calculations

6.22.4 Test cases

6.22.4.1 Period creation

Systems using time periods may allow users to create a new period by entering its beginning and ending date. A check may be made to ensure that the beginning date precedes the ending date. This check may fail when the time period spans Year 2000 rollover.

1999-12-27 to 2000-01-02 $\rightarrow$ 99-12-27 < 00-01-02

I Test objective

Verify that a time period may be created that crosses Year 2000 rollover.

II Test conditions

#	**Conditions**
1	Standard test system

III Procedure/results

#	**Test Procedure**	**Expected test results**
1	Attempt to create a time period starting Monday 1999-12-27 and ending Sunday 2000-01-02	Time period should be created

6.22.4.2 Grouping data by periods

I Test objective

Verify that weekly production data will be correctly summed when the weekly period spans the Year 2000 rollover.

II Test conditions

#	Conditions
1	Production tracking system
2	Weekly period for 1999-12-27 to 2000-01-02

III Procedure/results

#	Test Procedure	Expected test results
1	Enter the following daily production data. Date Qty. 1999-12-27 1 1999-12-28 10 1999-12-29 100 1999-12-30 1000 1999-12-31 10000 2000-01-01 100000 2000-01-02 1000000	
2	Check the total quantity for the period 1999-12-27 to 2000-01-02	The total should be 1111111

6.23 Queries, filters and data views

6.23.1 Definition

A higher-order function which takes a predicate and a list and returns those elements of the list for which the predicate is true.

6.23.2 Rationale

Filters operate by taking portions of dates and comparing values to similar portions of other dates. The ability to complete numerical comparisons on dates is essential to these elements. Using a two-character year representation in a query for records dated after the 6-digit date 00-01-01, for example, could return the entire database.

6.23.3 Test cases

6.23.3.1 Date comparisons

All logical operators will function normally in any combination. In all cases, dates after the Year 2000 will be considered greater than any date before it.

I Test objective

Ensure that <, and, > operate normally when filtering a list of files by dates.

II Test conditions

#	Conditions
1	Test period 1870 – 2030
2	A set of test files created on the following dates. a. 1870-01-01

	b. 1880-08-08 c. 1899-12-31 d. 1900-01-01 e. 1950-05-05 f. 1999-12-31 g. 2000-01-01 h. 2011-11-11 i. 2030-12-31

2772
2773 **III Procedure/results**

#	Test Procedure	Expected test results
1	Get a listing of file filtered by the following expressions	
2	date < 1880-08-08	a
3	date > 1880-08-08	c, d, e, f, g, h, i
4	date < 1950-05-05	a, b, c, d
5	date > 1950-05-05	f, g, h, i
6	date < 2011-11-11	a, b, c, d, e, f, g
7	date > 2011-11-11	i

2774

2775 **6.23.3.2 Using '<,>' operators**

2776 **I Test objective**

Verify that using inequalities in queries on databases that store dates both before and after the Year 2000 will function properly.

2777
2778 **II Test conditions**

#	Conditions
1	Database with records having the following date fields 1999-01-01 1999-09-09 1999-12-31 2000-01-01 2000-02-28 2000-02-29 2000-03-01 2000-12-31 2001-01-01 2035-01-01

2779
2780 **III Procedure/results**

#	Test Procedure	Expected test results
1	Perform query to find all values that are greater than 1999-12-10 and less than 2000-03-01	The query should return records with the dates 1999-12-31, 2000-01-01, 2000-02-28, and 2000-02-29.
2	Perform query to find all values that are less than 1999-12-10 and greater than 2001-01-01	The query should return records with the 1999-01-01, 1999-09-09, and 2035-01-01
3	Perform query to find all values that are less than 2000-01-01	The query should return records with the dates 1999-01-01, 1999-09-09, and 1999-12-31.

4	Perform other queries as necessary to check all logical operators used in the system.	Results will vary but should conform to standard logical operations.

6.24 Data recovery

6.24.1 Definition

Salvaging data stored on media, such as magnetic disks and tapes when normal access to the data is not available through normal system functions.

6.24.2 Rationale

File dates and times may be used to limit the number of files targeted by a recovery element. If the specified range is open-ended, it might cause a great deal of unnecessary data to be retrieved. If the recovery element is unable to distinguish the century part of the file date, a request to recover all files deleted after 00-02-01 (YY-MM-DD) might result in an attempt to restore any file deleted after 1900-02-01 (YYYY-MM-DD) along with the intended files.

6.24.3 Related elements

6.24.4 Test cases

6.24.4.1 Selection of files to retrieve

I Test objective

Verify that files deleted after Year 2000 rollover can be distinguished from pre-Year 2000 files when recovering data.

II Test conditions

#	**Conditions**
1	System date set to pre-Year 2000 rollover date

III Procedure/results

#	**Test Procedure**	**Expected test results**
1	Create and save a test file called “1999” then delete it.	
2	Change the system date to 2000-01-10.	
3	Create and save a test file called “2000” then delete it.	
4	Immediately use the recovery system element to retrieve files dated after 2000-01-01.	Only the file “2000” should be recovered. Note: If other testing has caused files to be created and deleted after 2000-01-01 they may also be recovered. No files dated prior to Year 2000 rollover should be recovered.

6.24.4.2 Database recovery using transaction logs and a prior image

This test ensures that a database which operates across the Year 2000 rollover and subsequently fails can be recovered by using an image taken before the Year 2000 rollover and the logs of transactions which have occurred since then. The logs will contain records of transactions occurring on both sides of the Year 2000 rollover. After the test is completed the recovered database should reflect all committed transactions recorded by those logs as well as all transactions recorded in the image taken prior to the Year 2000. The order of occurrence for multiple transactions affecting single database records should be maintained.

I Test objective

Ensure that the state of a post Year 2000 rollover database can be reconstructed from an image recorded prior to rollover by using transaction logs that cross the Year 2000 boundary.

II Test conditions

#	**Conditions**
1	A functioning database and Database Management System (DBMS) which processes transactions against that database and for which all changes are recorded in one or more logs.
2	A recovery system element for the DBMS.
3	System date set sufficiently prior to 1999-12-31 that the procedure can be accomplished before the date rolls over to 2000-01-01.

III Procedure/results

#	**Test procedure**	**Expected test results**
1	Run the application to process a few transactions against the database.	Database records should be updated normally. All changes should be logged correctly.
2	Create an image of the database at some point in time prior to the Year 2000 transition.	The image should represent the state of the database at the time the image was taken.
3	Continue running the application. Transactions should be processed that make changes to the database both before and after Year 2000 rollover. Resetting the system date if required.	The database should be updated normally. All changes should be logged correctly.
4	Stop, terminate or crash the database in mid-transaction.	The database should now require recovery due to an uncertainty of state.
5	Run the recovery procedure appropriate for the database, using Image created in step 2 and the logs of transactions that have occurred since the image was taken.	All database records updated by transactions committed since the last image was taken should be properly recovered. Check each changed database record to ensure that order of occurrence has been preserved where a single record is affected by multiple transactions.

6.25 Bridge testing

6.25.1 Definition

A bridge is a computer program operating between systems or parts of a system, that receives date information in one format and converts it to another format.

6.25.2 Rationale

When systems having several components are modified to become Year 2000 compliant, typically not all can be made compliant at once. As programs are modified, special bridges must be established to ensure that compliant elements do not cause non-compliant elements to fail, and vice versa. The use of bridges may also be a deliberate design choice, particularly for systems having large databases for which field expansion is not chosen as an option. These "bridges" between system elements must be tested to ensure they process dates correctly.

6.25.3 Test cases

6.25.3.1 Bridge date conversion

The bridge program must be able to correctly process all of the date formats that are used by the data it processes through its Year 2000 time horizon.

I Test objective

Verify that the bridge is able to accept and correctly format the outputs of all date formats likely to be used by the data it processes in a Year 2000 environment. It is assumed that the bridge will convert two digit dates to four digit dates, and vice versa. I.e., the bridge is bi-directional

II Test conditions

#	**Conditions**
1	A file of data to be converted by the bridge has data in two digit years. The file contains dates in the format used by the application. Consider the following date format possibilities:
2	A file of data to be converted by the bridge has data in four digit years. Consider the following date format possibilities:

Input Date	Four Digit		Two Digit	
	Format	Input/Output	Format	Input/Output
1999-12-31	MM/DD/YYYY	12/31/1999	MM/DD/YY	12/31/99
2000-12-31	MM-DD-YYYY	12-31-2000	MM-DD-YY	12-31-00
1999-12-31	MM.DD.YYYY	12.31.1999	MM.DD.YY	12.31.99
2000-12-31	YYYY/MM/DD	2000/12/31	YY/MM/DD	00/12/31
1999-12-31	YYYY-MM-DD	1999-12-31	YY-MM-DD	99-12-31
2000-12-31	YYYY.MM.DD	2000.12.31	YY.MM.DD	00.12.31
1999-12-31	DDMMYYYY	31121999	DDMMYY	311299
2000-12-31	YYYYMMDD	20001231	YYMMDD	001231

III Procedure/results

#	Test Procedure	Expected test results
1	Execute the bridge program using the appropriate file of date-data.	
2	Inspect the converted data.	The converted dates should match the expected date format results shown in the table above.

6.25.3.2 High risk date processing

The bridge should be able to correctly process all high-risk dates that may be included as data for conversion.

I Test objective

Verify that the bridge correctly processes all high-risk dates included in data to be converted.

High risk dates	
Four Digits	**Two Digits**
1999-01-01	99-01-01
1999-09-09	99-09-09
1999-12-31	99-12-31
2000-01-01	00-01-01
2000-12-31	00-12-31
2000-02-28	00-02-28
2001-01-01	01-01-01
2000-02-29	00-02-29
2001-02-29*	01-02-29*

* This date may produce an error if the bridge is responsible for detecting non-existent dates. If non-existent dates are detected in another system element the bridge may be required to accurately convert these dates so that they may continue to trigger error detection in other modules.

II Test conditions

#	Conditions
1	A bridge is prepared to accept data for conversion.
2	The file to be converted contains the high-risk date-data shown below.

III Procedure/results

#	Test Procedure	Expected test results
1	Execute the bridge program.	
2	Inspect the dates that are output	Output dates should match the expected results noted in the table above.

6.25.3.3 Bridge date window functionality

When date windowing algorithms are used in a bridge program to determine the correct century digits for a two-digit year, the windowing function should be tested.

I Test objective

Verify that two digit dates are correctly converted to four digit dates when windowing techniques are used.

II Test conditions

#	**Conditions**
1	A bridge is prepared to accept a data file for conversion.
2	Type of windowing implemented in the bridge is understood (fixed, movable, sliding).
3	Window boundaries are identified. These boundaries are unique to each application.
4	A data file is prepared containing dates (two or four digits) that are immediately before each boundary, on each boundary, and beyond each boundary.

III Procedure/results

#	**Test Procedure**	**Expected test results**
1	Execute the bridge program on the test data file.	
2	Inspect the resulting converted dates.	Those dates that were within or equal to the window boundaries should have "19" as the century digits and be processed accordingly. Dates that were outside the window boundaries should have the "20" century digits and be processed accordingly.

6.25.3.4 Database conversion

I Test objective

Verify that a function that converts a database for use by an application that works within a date window will filter out records with dates outside that window.

II Test conditions

#	**Conditions**
1	Conversion routine with pivot year set to 1960
2	Target application
3	Database that uses 8-character dates. 1900-01-01 1959-12-31 1960-01-01 1999-12-31 2000-01-01 2059-12-31 2060-01-01 2999-09-09

III Procedure/results

#	**Test Procedure**	**Expected test results**
1	Run conversion utility on the database and save the result in B.	File B should be created.
2	Open database B and view its content.	The date fields in database B should contain.

		1960-01-01 1999-12-31 2000-01-01 2059-12-31

6.26 Sorting

6.26.1 Definition

To arrange data by value.

6.26.2 Rationale

Sorting system elements rely on their ability to find truth values for the functions '>, =, <' when comparing date values. In some cases, a sort routine may interpret dates after the Year 2000 rollover as prior to 1999. An ascending order sort of data that spans the Year 2000 rollover might return a listing starting at 2000-01-01 and ending at 1999-12-31, with the most recent and the oldest data lost somewhere in the middle.

6.26.3 Related elements

Databases, Parsing, Filtering, Merge

6.26.4 Test cases

6.26.4.1 Data file sorted by date

Sorting a data file by ascending date should produce a sequence of records in which the earliest record dated in the Year 2000 should immediately follow the latest record for 1999 and all other records are be in order within their century.

I Test objective

Verify that sorting a data file by date produces a properly ordered list when the date field contains dates on both sides of the 1999 to 2000 boundary.

II Test conditions

#	**Conditions**
1	Application using a data file with records having the following date fields. 2000-01-01 1900-01-01 1925-07-04 2022-02-02 2029-12-31 1999-09-09 1999-12-31

III Procedure/results

#	Test Procedure	Expected test results
1	Sort the data file in ascending order by date. Display and/or print the results.	The dates should be displayed and/or printed in the following order. 1900-01-01 1925-07-04 1999-09-09 1999-12-31 2000-01-01 2022-02-02 2029-12-31
2	Sort the data file in descending order by date. Display and/or print the results.	The dates should be displayed and/or printed in the following order. 2029-12-31 2022-02-02 2000-01-01 1999-12-31 1999-09-09 1925-07-04 1900-01-01

6.26.4.2 Ordering a table

I Test objective

Verify that a printed list of tasks can be ordered by task end date when the project duration will cross the 1999 to 2000 boundary.

II Test conditions

#	Conditions
1	An application for project scheduling and a project with tasks that have the following dates. Start date End date 1999-12-06 1999-12-31 1999-12-15 2000-01-01 1999-12-14 2000-01-05 1999-12-29 2000-01-04 2000-01-01 2000-01-03 2000-01-01 2001-01-01

III Procedure/results

#	Test Procedure	Expected test results
1	Open the project in the application. Create a report with an ascending order list based on task completion Display and/or print the report.	The tasks should be listed in the following order. 1999-12-06 1999-12-31 1999-12-15 2000-01-01 2000-01-01 2000-01-03 1999-12-29 2000-01-04 1999-12-14 2000-01-05 2000-01-01 2001-01-01

6.27 User Interface (Input and Output)

6.27.1 Definition

This pertains to the means by which the user sends data into a system or system element and also by which the user interprets data from the same system.

6.27.2 Rationale

A user needs to be able to intuitively and efficiently enter data and interpret date-data from a system or system element. If date-data shortcuts are designed into the interface, a means should be available to unambiguously identify the data.
If a six-digit date format, (YY-MM-DD) is included to reduce keystrokes an accompanying note may be necessary to identify the range of possible dates. On-line help systems may need a review in order to document new screen and output formats.

6.27.3 Examples:

GUI's
Screen text
Printed reports

6.27.4 Related elements

6.27.5 Test cases

6.27.5.1 Display four digit year

I Test objective

Ensure that in time card screen form, start-of -period and end–of-period dates display in YYYY-MM-DD format.

II Test conditions

#	**Conditions**
1	System time set to 1999-12-27
2	Weekly time periods setup for 1999-12-26 to 2000-01-09

III Procedure/results

#	**Test Procedure**	**Expected test results**
1	Create time cards for both periods, and check the displayed dates	The cards are dated as follows. 1999-12-26 to 2000-01-01 2000-01-02 to 2000-01-09

6.28 Date format

6.28.1 Definition

An ordered arrangement of symbols, or data values used to define a day within a specified calendar system.

6.28.2 Rationale

Date formats vary depending on local custom. It is important that all date formats accepted by an application continue to be accepted and processed validly after Year 2000 remediation. Test cases in this section are designed to demonstrate that these date formats continue to work correctly.

6.28.3 Related elements - Globalization/internationalization

6.28.4 Test cases

6.28.4.1 Time/date input/output

I Test objective

Verify that all time/date-input formats available for the currently selected country continue to be accepted by the application in the Year 2000.

II Test conditions

#	Conditions
1	A remediated application with selectable date/time formats

III Procedure/results

#	Test Procedure	Expected test results
1	Set the application to its native date format	
2	Enter a series of test dates spanning the Year 2000 transition	All dates should be stored and displayed correctly
3	Select different date format.	
4	Display the previously entered dates	All dates entered in step 2 are correctly displayed in the new format.
5	Enter a second series of test dates spanning the Year 2000 transition	All dates should be stored and displayed correctly
6	Display the previously entered dates	All dates entered in step 2 are correctly displayed in the new format.
7	Repeat steps 3 through 6 for all other supported formats.	

6.28.4.2 Reports

I Test objective

Ensure that report date stamps will be correct after Year 2000 rollover.

II Test conditions

#	Conditions
1	System Date set to 2000-01-01

III Procedure/results

#	Test Procedure	Expected test results
1	Print a report and check it's date.	The date should be 2000-01-01
2	Change the system date to the last day of the test period. Print a	The date should be the same as the system date

	report and check its date.	
3	Repeat steps 1 and 2 for each supported date format.	The same results as 1 and 2 for each supported date format.

Annex - A Search strings (informative)

Certain strings are likely to contain date sensitive information for parsing. These could be in indexes, Table sizes, Data Dictionaries, Sort Routines, Filter routines, Date variables or documentation. This list is not to be considered all-inclusive. Less standard variable names may also have been used. In addition, formulas may have been use to convert to other formats that would have their own sets of identifiers. Searches for variable names can only be used as a starting point in finding affected system elements. Both functions called using theses parameters and functions containing the strings are suspect.

ANNIV	DD	PAYROLL
ANNIVERSARY	DDDYY	PROMOTION
AS-OF	DDMMYY	RETIRE
ASAP	DFDT	RELEASE
ASOF	DIFFDATE	START
BEG	DOB	T-O-D
BEGIN	DOH	TERM
BGN	DTE	TIME
BIRTH	DT	TIME-STAMP
CCYY	END	TIMEDATE
COMMISSION	ENLIST	TIMESTAMP
CYYDDD	ENLISTMENT	THISDATE
CYYDDMM	EXP	TOD
CYYMMDD	EXPIRE	TSTAMP
CURR	HIRE	WEEK
CURRENT	MDY	WEEKDAY
DA	MMDDYY	YEAR
DAT	MMM	YMD
DATE	MMYY	YR
DAY	MO	YY
DEATH	MON	YYDDD
DECOMMISSION	MONTH	YYMMDD
DEMOTION	PAROLL	

Annex - B Dates (informative)

B.1 Leap years

Most tests should be conducted in test environments that include some leap years. The addition of an extra day in the year will affect most date-related calculations if their interval encompasses February 29th. Date to day of week conversions will also be affected for days after Feb 29th.

B.2 Flags 01-01-99 - 12-31-00

All or some of these dates may be used to flag special logic functions. It is impractical to attempt to run a comprehensive set of tests on every day over a two-year period. Test should generally be run on the first and last days of 1999 and 2000 as well as any dates found during the upgrade process.

This problem may occur in data as well. Some professions have used 01-01-00 to indicate a date that is unknown to circumvent electronic forms that have required dates. As systems are upgraded to accept dates after the Year 2000 rollover these entries may take on meanings they were not intended to have.

B.3 Year 2000 rollover 12-31-1999 - 01-01-2000

Many processes compare date-data to system time or to other data. These functions should be tested with combinations of dates that span the Year 2000 rollover. These dates should include test conditions designed to produce errors such as negative time spans. In this way a review of error checking is done as well as a check of normal operation.

B.4 Counter rollover

System clocks may be based on binary counters that will eventually reach an upper limit related to the number of bits in the counters representation. The date 01-19-2038 - 03:14:08 has been identified as a common date for system clocks to rollover. It is based on a 32-bit counter interpreted as a signed integer value representing seconds from the beginning of 1970. Some systems may have different start times, use unsigned values or a different number of bits, that could change the actual rollover date. The above test date is a good place to start but a serious investigation into the operation of a particular system clock is necessary to be sure you have the proper date.

Annex - C Example archive documentation (informative)

The following items are considered essential to all Test Methodology Reports. The level of detail necessary in each individual area may vary.

As in the case of testing procedures themselves, appropriate documentation will vary depending on the organization involved, the level of risk of failure, the level of testing effort, the level of remediation, operational and time constraints, and many other factors. A large organization conducting comprehensive remediation and testing of a global network should keep correspondingly comprehensive documentation. A sole proprietor who determines to replace an aging computer rather than to seek to determine its compliance may need much less documentation. The following areas of documentation should be considered in light of the overall circumstances of an organization's Year 2000 effort.

Documentation should be maintained to assure that test methodologies and results can be understood when referenced in the future.

C.1 Overall organization and responsibilities for Year 2000 compliance

Inventory of hardware, firmware, and software

- Source, model, version, and other identifying information
- Vendors, suppliers, maintenance agreements
- Interfaces with other system elements
- Compliance, testing, and remediation information from available sources, including published standards and procedures, vendor correspondence, documentation and reports, print and internet materials, notes of verbal communications with vendors, consultants, users

Risk evaluation

- Potential business impact on organization and customers
- Potential public impact, harm to persons or property
- Triage and alternatives
- Testing and remediation capabilities, including time and cost constraints and availability of testing resources and personnel
- Conclusions: priorities, overall plan and scheduling for testing

Testing

- Specification of system hardware, firmware, software
- Test environment: interfaces, drivers, communications
- Identify and record any differences between the expected production environment and the test environment.
- Assumptions about testing environment, procedures and data
- Test procedures selected based on inventory and risk evaluation
- Test data used
- Report of results, including pass/fail/NA/Not Performed
- Identification of errors, exceptions, deviations from expected input or output, performance degradation
- Time and resources required for test

Remediation

- Corrective action taken and results
- Corrective action recommended but not taken
- Compatibility issues
- Additional testing recommended in light of results
- Time and resources required for remediation

Contingency planning

- Identification of potential failure contingencies and alternatives in light of status of testing and remediation efforts

- Triage to determine appropriate nature of contingency planning efforts in light of risk evaluation
- Plans for responding to failure contingencies

C.2 Specification of system under test

- A description of the software package under test.
- List of all third party component software
- Configuration information

C.3 Test environment specification

Include the hardware, types of networks and peripheral devices used.

C.4 Summary of general test methods and strategies

The use of module as opposed to system testing, environment generators, automated testing or other testing methods should be noted.

C.5 List of areas tested

- Discription of area tested
- Clear designation of pass or fail

C.6 List of areas not tested

C.7 Copy of all list cases executed against individual builds

C.8 Upper-bound / lower-bound dates

There may be several ranges for different types of data.

C.9 Exception logic/ error handling

A description of the conditions known to cause exceptions and errors related to date handling along with sequence of events followed when the exception or error occurs.

C.10 Input/output

Any deviation from accepted standards such as abnormal date formats or the need to pass additional information such as pivot dates should be noted.

C.11 Remediations

- Description of issues found
- Type of remediation used

C.12 Compatibility

- Products tested as compatible
- Known compatibility issues

Annex - D Alternative testing methodology (informative)

D.1 Alternatives to comprehensive testing.

The following tables describe alternative testing methods pertaining to levels of risk and levels of testing effort.

D.1.1 Levels of Risk

In order to prioritize a large number of applications, criteria will have to be established to group systems into risk categories. An example of a risk hierarchy is shown below.

Table: Levels of Risk

Level of Risk	Risk Manifestation
Non-Critical	No impact
Minor	No inconvenience
	Some inconvenience
	Customers notice problem
Moderate	Workarounds exist
	Workarounds needed
	Customers will be affected
Major	Major organizational impact; recovery plans exist
	Major organizational impact; no recovery plans exist

D.1.2 Levels of testing effort

The greater the number of steps between no testing and comprehensive testing that can be identified, the more flexibility an organization has in meeting its Year 2000 testing challenge with limited resources. These testing steps can be associated with the levels of risk described above.

A key issue in determining the levels of testing effort is the degree of system knowledge required to conduct testing. In many cases, detailed system knowledge is not readily available and would require significant work to capture.

The following table shows an example of a testing hierarchy that goes from no testing effort to great testing effort.

Table: Levels of testing effort

Level of test effort	Knowledge of system required	Testing technique
1	None	**No testing required.** Wait for possible failure and react to it when it occurs.
2	None	**Regression testing.** Compile code (if applicable) after Year 2000 related code modification.
	None	**Regression testing.** Verify that no pre-existing system functionality was lost after Year 2000 related code modification using production data and file comparison techniques. (*See more detailed discussion of techniques below.)*
	None	**Regression testing.** Use existing regression testing test cases or test beds to verify that no functionality was lost after code modification.

3	None	**Advance data dates.** Using production data, advance input dates to beyond 2000. Inspect resultant output dates for format and coherency.
4	None	**Age data 28 Years.** For systems using calendar dates, capture production related data inputs and outputs. Copy and modify the input data so that 28 years is added to each date. Compare the aged output file to the original output file. The dates should be identical except for the year. Automated tools can be used for this purpose.
5	None	**High risk date Set- No expected results.** Add a series of inputs that include high-risk dates. Inspect resulting output dates for format and coherence.
6	None	**Date simulators.** Use a copy of production data in a computer system in which a date simulator has been placed. The simulator will return a value for any date call made by the application program as a date beyond the Year 2000.
7	Some	**Key components.** In a system composed of many parts, identify key components such as conversion routines and test them comprehensively.
8	None	**Future environment.** If a separate system can be established in which the system clock has been advanced to beyond the Year 2000, place a copy of the application to be tested in the environment and execute the program. Inspect the output dates for format and coherence.
	None	**Future environment.** Test the application in the future environment using data aged to beyond the Year 2000. Inspect the output dates for format and coherence.
9	Moderate	**High risk dates - expected results.** Prepare a set of inputs that include high risk dates. Determine their expected output dates. Compare actual to expected results.
10	High	**Comprehensive testing.** Implement a structured testing methodology to ensure that all aspects of a system's functionality are tested. Compare actual to expected results.

D.1.2.1 File comparison / file aging techniques

A commonly used test technique that avoids the problem of having to understand the internal logic of a system is to compare production output before it is modified to its output after it is modified, when both are aged. For example, consider the following test steps:

1. Copy the production input and output of a system at a given time.
2. Modify the code of the system for Year 2000 compliance.
3. Run the original production input from step 1 through the modified code. The resultant output should be identical to the original output in step 1. This is classic regression testing.

4. Take a copy of the input file in step 1 and age it so the dates are all beyond the Year 2000. Run the file through the modified code and capture the aged output.
5. Take a copy of the original output file in step one and age the dates in the same way that the input file in step 4 was aged.
6. Compare the aged output in step 4 to the aged output in step 5. They should be identical except for the first two digits of the year. (On some file comparison tools these digits can be masked so that the other parts of the dates can be compared automatically.)

D1.1.2.2 Shortcomings in the use of production data to test

While the use of production data to test in an automated fashion has the appeal of being less difficult to accomplish than more traditional structured testing techniques, there are a number of risks associated with its use. Production data is not likely to contain Year 2000 high-risk dates. When production data is aged for testing it will not have dates likely to be acted upon as error codes, leap year dates or other dates at the boundaries of the system's date domain. If only production data is used, the impacts of these other dates will not be tested. There is a corresponding risk that should they be used in the future, the system may fail.

Annex - E Coverage overview (informative)

A comprehensive description of coverage analysis is beyond the scope of this document. If the Year 2000 testing staff is unfamiliar with these techniques there are many books on general testing procedures containing detailed descriptions of testing procedures. The purpose of this section is to give information about how these procedures can be applied to a Year 2000 project.

E.1 Level of testing

No reasonable amount of testing can guarantee that a non-trivial program will operate according to its specification. There is a point in every testing process when the cost of finding new errors becomes greater than the expected losses an undiscovered issue might cause. At the onset of the testing process it can be expected large numbers of errors exist, making them relatively easy to find. As testing continues the number of errors will decrease and those errors found are likely to be in more obscure paths. The number of hours required finding each new error should increase in a roughly exponential curve. The cost of continued testing to discover remaining errors eventually becomes greater than the cost of leaving the errors undiscovered.

E.2 Coverage analysis

A line by line analysis of code can be used to analyze the logical paths, branches, statements or the variable usage in part or all of a system’s code. This analysis can be done by hand or through the use of software tools designed for the purpose. When the analysis is complete the results can be used in the creation of test cases designed to ensure that all areas of the code under test will be exercised.

In a Year 2000 project it is often the case that a logical path that addresses special circumstances related to Year 2000 dates, doesn’t exist in the code. This may make it possible to exercise every decision, statement and variable in existing code without revealing that additional code is necessary to process Year 2000 dates.

Coverage analysis can be useful in regression testing of remediated systems to encourage an even distribution of testing effort, or it can be used to analyze code changes to promote thorough testing of logical branches added during remediation.

One method of coverage analysis involves tracking definitions, assignments and uses of variables within a piece of code. In this method a relation is formed between points in the code where a variable is defined or assigned a new value and points where the same variable is used as a source value for an operation. Def/Use or DU-pairs are formed when a logical path exists between these points. Test cases are then written that exercise each of the DU-pairs. As with all coverage analysis methods the process can be very time consuming. However in Year 2000 testing program the creation of DU-pairs could be limited to variables used for date processing thus providing a natural means of directing the analysis toward Year 2000 issues. A possible problem with this approach is that some date information may be stored in variables that are not easily recognizable, making it possible they will be left out of the analysis. It is also possible that errors introduced during the remediation process may not be date related, so normal regression testing will also be necessary.

E.3 Functional analysis

Functional analysis uses the planned functionality of the system as a guide to ensure all functions of the system are tested. The functionality of the system is described in the specification or documentation of the system, allowing test cases to be created without access to the systems code. The process involves breaking down the functions of the program into Equivalence Partitions. An Equivalence Partition tests a single series of operations or procedures, preformed by the system. A single function may perform many procedures depending on its input data.

For example consider an inventory system which includes a function to list all packages that have an expiration date prior to the current date. Two classes of test cases for this system element might be created to ensure:

- Correct packages are listed when the system date is before Year 2000 rollover.
- Correct packages are listed when the system date is after Year 2000 rollover.

But each class might require several operations be performed depending on the date-data tracked. Each of these broad classes can be split into several smaller classes. The first class might include sub classes:

- Correct packages are listed when the system date is before Year 2000 rollover and some packages expiration dates are before the start of the system valid date interval.

- Correct packages are listed when the system date is before Year 2000 rollover and all packages expiration dates are between the start of the system valid date interval and Year 2000 rollover.

- Correct packages are listed when the system date is before Year 2000 rollover and all packages expiration dates are within the system valid date interval but some are before Year 2000 rollover while others are after.

- Correct packages are listed when the system date is before Year 2000 rollover and all packages expiration dates are between the start of the system valid date interval and Year 2000 rollover.

- Correct packages are listed when the system date is after Year 2000 rollover and some packages expiration dates are after the end of the system valid date interval.

The process of breaking down the function into smaller classes should be continued until it can be reasonably expected that any test case written to test the class, tests for the same error. At this point a single test case should be written for each class.

Deciding when a class is homogeneous relies on the judgement of the test case author and his or her knowledge of the programming practices that affect how dates may be processed within the system. Most books that describe this technique will provide a list of thing to look for when creating equivalence classes. Boundaries and ranges related to dates are of particular importance in Year 2000 testing. Boundaries may be affected by the valid date intervals for each date format processed, Year 2000 rollover, leap year transitions and other system specific issues. Since the transition to Year 2000 may not have been considered when the system was specified some date ranges may not have been specified. In these cases it may be necessary to select a target valid date range and test functionality over that range.

Annex - F Informative references (informative)

1003.3 IEEE Standard for Information Technology - Requirements and Guidelines for Test Methods Specifications and Test Method Implementations for Measuring Conformance to POSIX Standards

610.12 IEEE Standard glossary of Software Engineering Terminology

829-1998 IEEE Standard for Software Test Documentation (reaffirmed 1991)

1008-1987 IEEE Standard for Software Unit testing

1012-1986 ANSI/IEEE Standard for Software Verification and Validation Plans

ANSI X3.30 - Formatting Date Data